

1-68/2

ST-JC
8-6-97

~~MICROFICHE~~ APPENDIX

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bglobal.h"
#include "vg_error.h"
#include "bparallel.h"
#include "stddevlp.h"
#ifdef _SEQUENT_
#include <sys/tmp_ctl.h>
#endif

EXEC SQL BEGIN DECLARE SECTION;
EXEC SQL END DECLARE SECTION;
#undef SQLCA_STORAGE_CLASS
EXEC SQL INCLUDE SQLCA.H;

int get_distribution(struct segment_struct **segment_list,
                    char *market,
                    long number_of_segments,
                    char *dynamic_load,
                    char *start_account,
                    char *end_account)
{
    EXEC SQL BEGIN DECLARE SECTION;
    char oacct_nr[11];
    VARCHAR ostart_account[10];
    VARCHAR oend_account[10];
    VARCHAR omarket[3];
    long orownum=0;
    long ocnt=0;
    long ototal_cust_count=0;
    long ototal_account_count=0;
    long osegment_size=0;
    EXEC SQL END DECLARE SECTION;

    struct segment_struct *segment_start=(struct segment_struct *)NULL;
    struct segment_struct *segment_last=(struct segment_struct *)NULL;
    struct segment_struct *segment_cur=(struct segment_struct *)NULL;
    struct segment_struct *segment_end=(struct segment_struct *)NULL;

    BOOLEAN error = FALSE; /* error flag */
    BOOLEAN first = TRUE; /* first account flag */
    long tot_cust_chk=0; /* count custs in segments */
    int index=0; /* count segments as produced */
    int indexa=0; /* count accounts as produced */
    int indexa_adj=0; /* count aggr overflow for segment */
    int segment_count=0; /* count segments as produced */
    long temp_acct_number=0;
    char segment_start_acct[11];
    char last_acct_nr[11];
    char segment_start_npa[4];
    char segment_start_str[8];
    long segment_start_num;
    long segment_mod=0;
    long distributor=0;
    long dist_adjust=0; /* Compensate for remainder after last segment */
    char line[80];
    FILE *fp; /* Static load file pointer */
    char tmp_err_buf[80]; /* used for formatted error statements */

    vput(&omarket,market);
    vput(&ostart_account,start_account);
    vput(&oend_account,end_account);
    memset(segment_start_acct,NULL,sizeof(segment_start_acct));

```

```

memset(last_acct_nr, NULL, sizeof(last_acct_nr));

if(dynamic_load[0] == '1')
{
    /* These queries assume pending accounts are not present in DB */

    EXEC SQL
        SELECT COUNT(account_nr)
            INTO :ototal_account_count
            FROM BILL_INFO
            WHERE MARKET = :omarket
              AND (ACCOUNT_NR BETWEEN :ostart_account AND :oend_account);

    EXEC SQL
        SELECT COUNT(account_nr)
            INTO :ototal_cust_count
            FROM CUST_INFO
            WHERE MARKET = :omarket
              AND ((aggr != 'A')
                 AND (ACCOUNT_NR BETWEEN :ostart_account AND :oend_account))
              OR (PARENT_ACCT BETWEEN :ostart_account AND :oend_account));

    if((ototal_cust_count == 0) || (ototal_account_count == 0))
    {
        error = TRUE;
        error_handler("get_distribution", UNKNOWN,
                     "Need to specify an account range "
                     "encompassing actual accounts.");
        return(error);
    }
    else if(number_of_segments > 0)
    {
        /* Must kludge this until able to bill aggs across batches */
        if(ototal_account_count/number_of_segments >= 0)
        {
            osegment_size = ototal_cust_count/number_of_segments;
            /* mod is the overflow to be evenly distributed */
            segment_mod = ototal_cust_count % number_of_segments;

            /* protect for divide by zero */
            if(segment_mod != 0)
                distributor = number_of_segments/segment_mod;
            else distributor = 0;
        }
        else
        {
            osegment_size = 0;
        }

        if(osegment_size == 0)
        {
            /* don't run parallel if one account per segment */
            /* overhead is worse than sequential */
            osegment_size = 1;
            number_of_segments = 1;
            error_handler("get_distribution", UNKNOWN,
                         "Warning: Segment size < 1 account per ... "
                         "reset to one segment.");
        }
        /* If there are more segments than accounts */

        printf("start_account = %10.10s end_account = %10.10s "
              "num accts = %1d\n",
              start_account, end_account, ototal_account_count);
    }
}

```

```

printf("custs = tld seg size = tld num segs = tld "
      "mod = tld dist = tld\n",
      ototal_cust_count, osegment_size, number_of_segments,
      segment_mod, distributor);
}
else
{
    error = TRUE;
    error_handler("get_distribution", UNKNOWN,
                  "Number of segments cannot be zero.");
    return(error);
}

EXEC SQL DECLARE segments CURSOR FOR
SELECT NVL(PARENT_ACCT, ACCOUNT_NR)
FROM CUST_INFO
WHERE MARKET = :omarket
AND (NVL(PARENT_ACCT, ACCOUNT_NR)
     between :ostart_account AND :oend_account)
ORDER BY NVL(PARENT_ACCT, ACCOUNT_NR) ASC;

EXEC SQL OPEN segments;

if(sqlca.sqlcode != NOT_SQL_ERROR)
{
    error_handler("get_distribution", UNKNOWN, sqlca.sqlerrm.sqlerrmc);
}

while((sqlca.sqlcode == NOT_SQL_ERROR) && (!error))
{
    /* distribute extra accounts if more left in overflow (segment mod) and
    distributor indicates some segments get an extra account. */

    if((distributor != 0) && (segment_mod > 0) &&
        ((segment_count % distributor) == 0))
    {
        /* add an extra account to segment size */
        dist_adjust = 1;

        /* adjust so when extra accounts are depleted, no more extra segment
        space will be allocated */

        segment_mod--;
    }
    else dist_adjust = 0;

    /* Fetch another segment */
    while((sqlca.sqlcode == NOT_SQL_ERROR) &&
          (index < (osegment_size + dist_adjust)) &&
          (!error))
    {
        EXEC SQL FETCH segments INTO :oacct_nr;
        if((sqlca.sqlcode != NOT_SQL_ERROR) &&
            (sqlca.sqlcode != SQL_NOT_FOUND))
        {
            segment_start = (struct segment_struct *)NULL;
            error_handler("get_distribution", UNKNOWN,
                          sqlca.sqlerrm.sqlerrmc);
            error = TRUE;
        }
        /* error */
        else if(sqlca.sqlcode != SQL_NOT_FOUND)
        {
            /* Fetch at end throws off customer count */
            index++;
        }
    }
}

```

```

if(first)
{
    first = FALSE;
    memcpy(segment_start_acct,oacct_nr,10);
}

/* Just logging a count of accounts vs customers (actual) */
if(memcmp(oacct_nr,last_acct_nr,sizeof(oacct_nr)) != 0)
{
    indexa++;
    memcpy(last_acct_nr,oacct_nr,sizeof(oacct_nr));
}
if((indexa == 0) &&
    (memcmp(oacct_nr,last_acct_nr,sizeof(oacct_nr)) == 0))
{
    indexa_adj++;
}
}/* no error fetching next customer */
}/* While not segment limit */

/* allocate a list element (0th counts here) */
if((segment_count < number_of_segments) &&
    ((sqlca.sqlcode == SQL_NOT_FOUND) ||
    (sqlca.sqlcode == NOT_SQL_ERROR)))
{
    if ((segment_cur = (struct segment_struct *)
        malloc((unsigned int)sizeof(struct segment_struct)))
        != (struct segment_struct *)NULL)
    {
        segment_count++;

        /* Load the segment element */
        sprintf(segment_cur->rpt_file,"%3.3s %d",
            omarket.arr,segment_count);
        if(osegment_size > 1)
            memcpy(segment_cur->begin_acct,
                segment_start_acct,sizeof(oacct_nr));
        else
            memcpy(segment_cur->begin_acct,oacct_nr,
                sizeof(oacct_nr));
        segment_cur->begin_acct[10] = '\0';
        memcpy(segment_cur->end_acct,oacct_nr,sizeof(oacct_nr));
        segment_cur->end_acct[10] = '\0';
        sprintf(segment_cur->stdout_file,"%3.3s %d",
            omarket.arr,segment_count);
        segment_cur->segment_number = segment_count;
        segment_cur->process_id = 0;
        segment_cur->processor = 0;
        segment_cur->running = 0;
        segment_cur->row_num = 0;

        /* adjust customer count to reflect aggregates that went to previous segment */
        segment_cur->csize = index - indexa_adj;

        /* account count in this segment */
        segment_cur->asize = indexa;
        segment_cur->count = 0;
        segment_cur->complete = 0;
        segment_cur->link = (struct segment_struct *)NULL;

        /* if this is the first element then mark it as the head of the list */
        if (segment_start == (struct segment_struct *)NULL)
        {
            segment_start = segment_end = segment_cur;
        } /* if start of list */
        else
        {

```

```

/* adjust customer count in previous segment to reflect its aggr overflows */
    segment_end->csize += indexa_adj;
    tot_cust_chk += segment_end->csize;
    segment_end->link = segment_cur;
    segment_end = segment_cur;
} /* else not start of list */

/* Increment end account to use as start of next segment */
    sprintf(segment_start_npa, "%3.3s", segment_end->end_acct);
    sprintf(segment_start_str, "%7.7s",
        &segment_end->end_acct(3));
    segment_start_num = atol(segment_start_str);
    segment_start_num++;
    sprintf(segment_start_acct, "%3.3s%07ld",
        segment_start_npa,
        segment_start_num);

} /* if allocate list element */
else
{
    segment_start = (struct segment_struct *)NULL;
    error_handler("get_distribution", UNKNOWN,
        "memory allocation");
    error = TRUE;
} /* else malloc error */
}/* If fetch */
else if((segment_count >= number_of_segments) &&
    (sqlca.sqlcode != SQL_NOT_FOUND))
{
    if(memcmp(oacct_nr, last_acct_nr, sizeof(oacct_nr)) != 0)
    {
        sprintf(tmp_err_buf,
            "Out of segments and account %10.10s left.",
            oacct_nr);
        segment_start = (struct segment_struct *)NULL;
        error_handler("get_distribution", UNKNOWN, tmp_err_buf);
        error = TRUE;
    }
    else
    {
        segment_end->csize++;
        while((sqlca.sqlcode != SQL_NOT_FOUND))
        {
            if(memcmp(oacct_nr, last_acct_nr, sizeof(oacct_nr)) != 0)
            {
                sprintf(tmp_err_buf,
                    "Out of segments and account "
                    "%10.10s left.", oacct_nr);
                segment_start = (struct segment_struct *)NULL;
                error_handler("get_distribution", UNKNOWN,
                    tmp_err_buf);
                error = TRUE;
            }
            segment_end->csize++;
            EXEC SQL FETCH segments INTO :oacct_nr;
        }
    }
} /* error if out of segments and more accounts left */

/* reset index for next goround */
index = 0;
indexa = 0;
indexa_adj = 0;

}/* While more segments */

```

```

memcpy(segment_end->end_acct,end_account,10);

if(sqlca.sqlcode != SQL_NOT_FOUND)
{
    segment_start = (struct segment_struct *)NULL;
    error_handler("get_distribution",UNKNOWN,sqlca.sqlerrm.sqlerrmc);
    error = TRUE;
}/* Report error */

EXEC SQL CLOSE segments;
/* get last segments' customer allotment */
tot_cust_chk += segment_end->csize;

printf("%ld TOTAL IN SEGMENTS = %ld in db = %ld\n",
    segment_count,tot_cust_chk,tot_cust_count);
}
else
{
    if((fp = fopen("LOAD_BALANCE","r")) == NULL)
    {
        segment_start = (struct segment_struct *)NULL;
        error_handler("get_distribution",UNKNOWN,
            "Can't open LOAD_BALANCE file for "
            "segmenting information");
        error = TRUE;
    }
    else for(segment_count = 1;
        segment_count <= number_of_segments;
        segment_count++)
    {
        /* Load X number of segments (error if proper number not found) */
        if(fgets(line,80,fp) != (char)NULL)
        {
            if ((segment_cur = (struct segment_struct *)
                malloc((unsigned int)sizeof(struct segment_struct)))
                != (struct segment_struct *)NULL)
            {
                printf("STATIC_LOAD MALLOC\n");

                /* Load the segment element */
                sprintf(segment_cur->rpt_file,"%s_td",market,
                    segment_count);
                memcpy(segment_cur->begin_acct,line,10);
                segment_cur->begin_acct[10] = '\0';
                memcpy(segment_cur->end_acct,&line[11],10);
                segment_cur->end_acct[10] = '\0';
                sprintf(segment_cur->stdout_file,"%s_td",
                    market,segment_count);
                segment_cur->segment_number = segment_count;
                segment_cur->process_id = 0;
                segment_cur->processor = 0;
                segment_cur->running = 0;
                segment_cur->row_num = 0;
                segment_cur->csize = 0;
                segment_cur->asize = 0;
                segment_cur->count = 0;
                segment_cur->complete = 0;
                segment_cur->link = (struct segment_struct *)NULL;

                /* if this is the first element then mark it as the head of the list */
                if (segment_start == (struct segment_struct *)NULL)
                {
                    segment_start = segment_end = segment_cur;
                } /* if start of list */
            }
            else

```

```

    {
        segment_end->link = segment_cur;
        segment_end = segment_cur;
    } /* else not start of list */

} /* if allocate list element */
else
{
    segment_start = (struct segment_struct *)NULL;
    error_handler("get_distribution",UNKNOWN,
        "memory allocation");
    error = TRUE;
} /* else malloc error */
}/* If get segment line */
else
{
    segment_start = (struct segment_struct *)NULL;
    sprintf(line,"Can't get segment range entry %d of %d",
        segment_count,number_of_segments);
    error_handler("get_distribution",UNKNOWN,line);
    error = TRUE;
}
}/* for x segments */
}

/* Place starting address of segment list in caller's pointer */
*segment_list = segment_start;

return(error);
}/* End of get distribution */

```



```
#include "stddevip.h"
#include "bill_global.h"
#include "vg_error.h"
#include "par_man_proto.h"
```

```
EXEC SQL BEGIN DECLARE SECTION;
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA.H;
```

```
BOOLEAN get_executable(char *path, char *name)
```

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR opath[50];
    VARCHAR oname[20];
    EXEC SQL END DECLARE SECTION;

    BOOLEAN error = FALSE;

    EXEC SQL
        SELECT EXECUTABLE_PATH, EXECUTABLE_NAME
            INTO :opath, :oname
        FROM BILLING_PARAMETERS
        WHERE ROWNUM = 1;

    if (sqlca.sqlcode != NOT_SQL_ERROR)
    {
        error = TRUE;
        error_handler("get_executable.pc", ORACLESELECT,
            "selecting executable info");
    }

    vget(path, &opath);
    vget(name, &oname);

    return error;
}
```

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <osfcn.h>
#include <fcntl.h>
#include <sgtty.h>
#include <sys/resource.h>
#include <sys/signal.h>
#include <sys/stat.h>
#ifdef _SEQUENT_
#include <sys/tmpctl.h>
#endif
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <sys/vmstat.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include "time.h"
#include "bill_global.h"
#undef BOOLEAN
#include "stddevlp.h"
#include "vg_error.h"
#include "bparallel.h"

/* TEMP DEBUG */
char *a;
char *b;

struct mark_struct
{
    char remark[81];
    long seconds;
    long useconds;
};

#ifdef _SEQUENT_
extern "C" {
    char *shmat(int, void*, int);
    int shmget(key_t, int, int);
}
#endif

union {
    struct vm_tune *vmtune;
    unsigned long *procrss;
    bool_t onoff;
}argp;
#endif

struct par_perf_struct par_per;
struct seg_perf_struct seg_per;

void shmark_time(int remark_nr, mark_struct *time_array, int mark_number);
void fork_segment(segment_struct *segment,
    char arg_list[ARG_COUNT][MAX_ARG_SIZE],
    char *shmaddress, char *executable);

int main(int argc, char **argv)
{
    struct segment_struct *segment_list_start=(struct segment_struct *)NULL;
    struct segment_struct *segment_list=(struct segment_struct *)NULL;
    int error=0, finished=0;
    int affinity_err_adj=0, cpu_num=0, set_p=0, number_of_cpus=0;

```

```

int process_status=0,accounted_for=0,found=0,wait_count=0;
int previous_processor=0,index=0;
char market(4);
long number_of_segments=0;
long number_of_processes=0;
char arg_list[ARG_COUNT][MAX_ARG_SIZE];
char tmparg1(3);
char oracle_login(40);
char bill_date(11);
char commit_flag(2);
char override_flag(2);
char dynamic_load(2);
BOOLEAN reports_flag;
char tmp_err_str(80);
#ifdef _SEQUENT_
int process_group=0;
#else
pid_t process_group=0;
#endif

char tmpindex_err_str(80);
char start_account(11);
char end_account(11);
char billing_path(51);
char billing_name(21);
char full_billing_name(71);

/* Shared memory vars */
BOOLEAN shared=0;
key_t shbill_key=SHARED_MEM_KEY;
int shbill_id;
int shmflg=1;
char *shmaddress;
char *shmaddress_s;

struct mark_struct mark_time_arr(80);

pid_t current_pid=0;

sprintf(mark_time_arr(0).remark,"OVERALL ");
mark_time_arr(0).useconds = 0L;
mark_time_arr(0).seconds = 0L;

sprintf(mark_time_arr(1).remark,"LOAD BALANCE ");
mark_time_arr(1).useconds = 0L;
mark_time_arr(1).seconds = 0L;

sprintf(mark_time_arr(2).remark,"REPORT GENERATION ");
mark_time_arr(2).useconds = 0L;
mark_time_arr(2).seconds = 0L;

sprintf(mark_time_arr(3).remark,"THREAD FILE MERGE ");
mark_time_arr(3).useconds = 0L;
mark_time_arr(3).seconds = 0L;

setbuf(stdout,NULL);

/* Set process group so parallel manager (this program) is part of it. */
if((process_group = setpgid()) == -1)
{
    sprintf(tmp_err_str,
        "FATL: Unable to obtain process group id for this bill run");
    error_handler("par_bill.pc",UNKNOWN,tmpindex_err_str);
}
/* Validate command line arguments */
if(argc != 11)
{

```

```

fprintf(stderr,
    "Usage: par_bill market bill_date oracle_login "
    "commit_flag(0,1) override_flag(0,1) "
    "dynamic_load_flag(0,1) reports_flag(0,1) "
    "[segments] start end\n");
_exit(0);
}
else
{
    shmark_time(0,mark_time_arr,1);
    sprintf(market,"%s",argv[1]);
    sprintf(bill_date,"%s",argv[2]);
    sprintf(oracle_login,"%s",argv[3]);
    sprintf(commit_flag,"%s",argv[4]);
    sprintf(override_flag,"%s",argv[5]);
    sprintf(dynamic_load,"%s",argv[6]);
    reports_flag = atoi(argv[7]);

    number_of_cpus = get_cpus();
    printf("Number of cpus = %d\n",number_of_cpus);

    /* Allow user to assign segment list or set via available cpus */
    if((argc >= 9) && (argc != 10))
    {
        number_of_segments = atoi(argv[8]);
    }
    else
    {
        number_of_segments = (number_of_cpus - 1);
    }

    if(argc == 11)
    {
        printf("ARGS start = %10.10s end = %10.10s\n",argv[9],argv[10]);
        sprintf(start_account,"%s",argv[9]);
        sprintf(end_account,"%s",argv[10]);
    }
    else
    {
        sprintf(tmp_err_str,
            "This batch will bill every account for market %s",market);
        error_handler("par_bill.pc",UNKNOWN,tmp_err_str);
        strcpy(start_account,"0000000000");
        strcpy(end_account,"9999999999");
    }

    number_of_processes = number_of_segments;

    /* load command line arguments. */

    if ((oracleLogin(oracle_login,NULL)) != -1)
    {
        /* Allocate shared memory block for manager and threads */
        /* if not existing */
        while(!(!shared) && (!error))
        {

```

```

/* Allocate shared memory sufficient for parallel bill run */
shbill_id = shmget(shbill_key,
    (int)(sizeof(struct par_perf_struct) +
        ((60)*(sizeof(struct seg_perf_struct)))),
    (0666 | IPC_CREAT));

if(shbill_id == -1)
{
    error = TRUE;
    sprintf(tmp_err_str,
        "Shared memory allocation for %d: attempt failed.",
        shbill_key);
    error_handler("par_bill.pc",UNKNOWN,tmp_err_str);
    exit(0);
}/* Get new key if in use */
else
{
    shared = 1;
#ifdef _SEQUENT_
    shmaddress = shmat(shbill_id,0,0);
#else
    shmaddress = (char *)shmat(shbill_id,0,0);
#endif
    if((int)shmaddress == -1)
    {
        error = TRUE;
        sprintf(tmp_err_str,
            "shmat() had error attaching %d to data segment.",
            shbill_id);
        error_handler("par_bill.pc",UNKNOWN,tmp_err_str);
        exit(0);
    }
    else
    {
        par_per.segments = number_of_segments;
        par_per.status = 1;
        par_per.load_bal_time = 0;
        par_per.rpt_build_time = 0;
        par_per.rpt_merge_time = 0;
        memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));
    }

    /* allocate shared memory ok */
}/* Allocate shared memory for inter process communication */

if (error = get_executable(billing_path,billing_name))
{
    error_handler("par_bill.c",UNKNOWN,
        "Unable to find billing executable name");
    exit(0);
}
else
{
    sprintf(full_billing_name,"%s/%s",billing_path,billing_name);

printf("market = %3.3s nos = %ld nop = %ld error before distribution = %d\n",
    market,number_of_segments,number_of_processes,error);
printf("start = %10.10s end = %10.10s\n",
    start_account,end_account);

    seg_per.seg_bills = 0;
    seg_per.seg_accts = 0;
    seg_per.segment_number = 0;
    seg_per.process_id = 0;

```

```

seg_per.processor = 0;
seg_per.running = 0;
seg_per.complete = 0;
seg_per.slow_time = 0;
seg_per.fast_time = 0;
seg_per.last_acct_time = 0;
seg_per.last_cust_time = 0;
seg_per.elapsed_time = 0;
seg_per.total_time = 0;
seg_per.bill_count = 0;
seg_per.acct_count = 0;
memcpy(seg_per.last_account,"XXXXXXXXXX",10);
memcpy(seg_per.last_cust,"XXXXXXXXXX",10);

for(index = 1;index <= number_of_segments;index++)
{
    shaddress_s = (shaddress + (sizeof(struct par_perf_struct) +
                                ((index - 1) *
                                 sizeof(struct seg_perf_struct))));
    memcpy(shaddress_s,&seg_per,sizeof(struct seg_perf_struct));

}/* Initialize shared memory for each threagment. */

/* Get load distribution (processing segments) */
shmark_time(1,mark_time_arr,1);
error = get_distribution(&segment_list,
                        market,
                        number_of_segments,
                        dynamic_load,
                        start_account,
                        end_account);

shmark_time(1,mark_time_arr,2);
par_per.status = 2;
memcpy(shaddress,&par_per,sizeof(struct par_perf_struct));

segment_list_start = segment_list;
printf("error after distribution = %d\n",error);

/* Don't need database anymore. */
oracleLogout();

while(segment_list != (struct segment_struct *)NULL)
{
    printf("%s ",market);
    printf("%s ",segment_list->rpt_file);
    printf("%s ",oracle_login);
    printf("%s ",commit_flag);
    printf("%s ",override_flag);
    printf("%s ",dynamic_load);
    printf("%s ",bill_date);
    printf("%s ",segment_list->begin_acct);
    printf("%s ",segment_list->end_acct);
    printf("%s ",segment_list->stdout_file);
    printf("%d ",segment_list->segment_number);
    printf("%d ",segment_list->process_id);
    printf("%d ",segment_list->processor);
    printf("%d ",segment_list->running);
    printf("%d ",segment_list->complete);
    printf("%ld ",segment_list->csize);
    printf("%ld\n",segment_list->asize);

    seg_per.seg_bills = segment_list->csize;
    seg_per.seg_accts = segment_list->asize;
    shaddress_s =
        (shaddress + (sizeof(struct par_perf_struct) +

```

```

        ((segment_list->segment_number - 1) *
         sizeof(struct seg_perf_struct)));
memcpy(&seg_per, shmaddress_s, sizeof(struct seg_perf_struct));

segment_list = segment_list->link;

/* traverse */

segment_list = segment_list_start;

/* Fork X segments of the bill run and maintain that number
 * until entire segment list is completed.
 */

/* Set up non segment-specific argument list execution */
sprintf(arg_list[0], "%s", billing_name);
sprintf(arg_list[1], "%s", market);
sprintf(arg_list[3], "%s", oracle_login);
sprintf(arg_list[4], "%s", bill_date);
sprintf(arg_list[5], "%s", commit_flag);
sprintf(arg_list[6], "%s", override_flag);
if(number_of_segments == 1)
    sprintf(arg_list[7], "S");
else
    sprintf(arg_list[7], "P");
sprintf(arg_list[12], "%s", "");

for(index = 1; index <= number_of_processes; index++)
{
    /* create child process */
    fork_segment(segment_list, arg_list, shmaddress,
                full_billing_name);

    /* if successful fork, handle next segment in list */
    if(segment_list != (segment_struct *)NULL)
    {
        segment_list = segment_list->link;
    }
    else if(index != number_of_processes)
    {
        sprintf(tmp_err_str,
            "WARN: Exhausted segment list at %d before "
            "reaching last (%dth) segment.",
            index, number_of_processes);
        error_handler("par_bill.pc", UNKNOWN, tmp_err_str);
    }
    /* Make sure finished when list is exhausted. */

    printf("FORK\n");
}

/* end for x segments */

segment_list = segment_list_start;
while(segment_list != (struct segment_struct *)NULL)
{
    /* Put process ID into shared memory for this segment */
    shmaddress_s = (shmaddress + (sizeof(struct par_perf_struct) +
        ((index - 1) *
         sizeof(struct seg_perf_struct))));
    memcpy(&seg_per, shmaddress_s, sizeof(struct seg_perf_struct));
    seg_per.process_id = segment_list->process_id;
    printf("SHARED MEM PROCESS ID %d %d\n", seg_per.process_id,
        seg_per.segment_number);
    memcpy(shmaddress_s, &seg_per, (sizeof(struct seg_perf_struct)));
}

```

```

    segment_list = segment_list->link;
}/* traverse */

while(!finished)
{
    /* Monitor pids and fork as needed until segment_list exhausted */
    current_pid = waitpid(0,&process_status,0);
    if((current_pid != 0) && (current_pid != -1))
    {
        printf("good process_status = %d\n",process_status);

        /* Find segment and processor number of this process */
        /* for reporting. */
        segment_list = segment_list_start;
        found=0;
        index=0;
        while((segment_list != (struct segment_struct *)NULL) &&
            (!found))
        {
            if(segment_list->process_id == current_pid)
            {
                index = segment_list->segment_number;
                previous_processor = segment_list->processor;
                found=1;
            }
            else segment_list = segment_list->link;
        }/* while looking for segment that matches this pid */

        if(WIFEXITED(process_status) != 0)
        {
            printf("DETECTED NORMAL\n");
            if(WEXITSTATUS(process_status) == 0)
            {
                printf("DETECTED NO ERROR\n");
                /* If exit was ok, then fork another segment while more is left, accounting
                 * for segment just completed in the segment list.
                 */
                segment_list = segment_list_start;
                accounted_for = 0;
                while((!accounted_for) &&
                    (segment_list != (segment_struct *)NULL))
                {
                    /* Mark segment as completed */
                    if(current_pid == segment_list->process_id)
                    {
                        segment_list->complete = accounted_for = 1;
                        segment_list->running = 0;
                    }
                    else segment_list = segment_list->link;
                } /* Account for segment just completed */

                if(!accounted_for)
                {
                    sprintf(tmp_err_str,
                        "WARN: Process %d running segment ? "
                        "is unaccounted for.",
                        current_pid);
                    error_handler("par_bill.pc",UNKNOWN,tmp_err_str);
                }

                /* Find next segment to be executed */
                found=0;
                segment_list = segment_list_start;
                while ((segment_list !=
                    (struct segment_struct *)NULL) &&

```



```

        (!found))
    {
        if((segment_list->running == 0) &&
            (segment_list->complete == 0))
        {
            /* Fork another segment to replace completed one. */
            fork_segment(segment_list, arg_list, shmaddress,
                full_billing_name);

            sprintf(tmparg1, "pid created: %d",
                segment_list->process_id);
            printf("tmparg1 = %s\n", tmparg1);
            found = 1;
        }
        /* Fork a new segment */
        else segment_list = segment_list->link;
    }
    /* While looking for next segment to execute */

    if(!found)
    {
        finished = 1;
    }
    /* All segments are or were running. */
    /* Run manager is finished. */
}
/* If _exit(0) */
else
{
    printf("DETECTED ERROR\n");

/*
* If exited due to error, kill all other segments, report error, and die.
*/

    sprintf(tmp_err_str,
        "FATAL: Process %d running segment %d "
        "terminated with error.",
        current_pid, index);
    error_handler("par_bill.pc", UNKNOWN, tmp_err_str);
    par_per.status = -1;
    memcpy(shmaddress,
        &par_per, sizeof(struct par_perf_struct));
    seg_per.running = 0;
    shmaddress_s = (shmaddress +
        (sizeof(struct par_perf_struct) +
        ((segment_list->segment_number - 1) *
        sizeof(struct seg_perf_struct))));
    memcpy(shmaddress_s, &seg_per,
        sizeof(struct seg_perf_struct));
    kill(0, SIGKILL);
}
/* _exit(1) */
}
/* process terminated normally */
else if(WIFSIGNALED(process_status) != 0)
{
    printf("DETECTED KILL\n");
    /* Report that process was killed and kill */
    /* all others before exiting. */
    sprintf(tmp_err_str,
        "FATAL: Process %d running segment %d was "
        "killed by signal %d",
        current_pid, index, WTERMSIG(process_status));
    error_handler("par_bill.pc", UNKNOWN, tmp_err_str);
    par_per.status = -1;
    memcpy(shmaddress, &par_per, sizeof(struct par_perf_struct));
    seg_per.running = 0;
    shmaddress_s = (shmaddress +
        (sizeof(struct par_perf_struct) +
        ((segment_list->segment_number - 1) *
        sizeof(struct seg_perf_struct))));
    memcpy(shmaddress_s, &seg_per,
        sizeof(struct seg_perf_struct));

```

```

        kill(0,SIGKILL);
    }/* Killed by signal */
#ifdef _SEQUENT_
    else if(WIFCORESIG(process_status) != 0)
#else
    else if(WCOREDUMP(process_status) != 0)
#endif
    {
        printf("DETECTED CORE\n");
        sprintf(tmp_err_str,
            "FATL: Process %d running segment %d was "
            "killed by signal %d causing core dump.",
            current_pid,index,WTERMSIG(process_status));
        error_handler("par_bill.pc",UNKNOWN,tmp_err_str);
        par_per.status = -1;
        memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));
        seg_per.running = 0;
        shmaddress_s = (shmaddress +
            (sizeof(struct par_perf_struct) +
            ((segment_list->segment_number - 1) *
            sizeof(struct seg_perf_struct))));
        memcpy(shmaddress_s,&seg_per,
            sizeof(struct seg_perf_struct));
        kill(0,SIGKILL);
    }/* Core dump */
    else if(WSTOPSIG(process_status) != 0)
    {
        printf("DETECTED STOP\n");
        sprintf(tmp_err_str,
            "FATL: Process %d running segment %d was "
            "stopped by signal %d.",
            current_pid,index,WTERMSIG(process_status));
        error_handler("par_bill.pc",UNKNOWN,tmp_err_str);
        par_per.status = -1;
        memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));
        seg_per.running = 0;
        shmaddress_s = (shmaddress +
            (sizeof(struct par_perf_struct) +
            ((segment_list->segment_number - 1) *
            sizeof(struct seg_perf_struct))));
        memcpy(shmaddress_s,&seg_per,
            sizeof(struct seg_perf_struct));
        kill(0,SIGKILL);
    }/* Stop signal */
    else
    {
        printf("DETECTED UNKNOWN CONDITION\n");
        sprintf(tmp_err_str,
            "WARN: Process %d running segment %d "
            "affected by signal %d.",
            current_pid,index,WTERMSIG(process_status));
        error_handler("par_bill.pc",UNKNOWN,tmp_err_str);
        par_per.status = -1;
        memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));
        seg_per.running = 0;
        shmaddress_s = (shmaddress +
            (sizeof(struct par_perf_struct) +
            ((segment_list->segment_number - 1) *
            sizeof(struct seg_perf_struct))));
        memcpy(shmaddress_s,&seg_per,
            sizeof(struct seg_perf_struct));
        kill(0,SIGKILL);
    }/* Unknown signal */
    wait_count = 0;
}
else

```

```

{
    if(current_pid == -1)
    {
        printf("process_status = %d\n",process_status);
        sprintf(tmp_err_str,
            "WARN: monitor1: wait pid is finished. "
            "Parallel monitor1 is terminating.");
        error_handler("par_bill.pc",UNKNOWN,tmp_err_str);
        finished = 1;
    }/* wait pid error dump */
    else
    {
        printf("process_status = %d\n",process_status);
        wait_count++;
        sprintf(tmp_err_str,
            "WARN: monitor1: No status was returned.");
        error_handler("par_bill.pc",UNKNOWN,tmp_err_str);

        sleep(5);
        if(wait_count == MAX_WAIT) finished = 1;
    }/* wait pid error dump */
}/* Problems with wait pid */
} /* maintain X processes until all segments are completed */
printf("FINISHED MONITOR.\n");

finished = 0;
while(!finished)
{
    /* Monitor pids until all have completed without errors.*/
    /* removed no hang up WNOHANG so it should wait till */
    /* something happens */
    current_pid = waitpid(0,&process_status,0);
    if((current_pid != 0) && (current_pid != -1))
    {
        printf("good process_status = %d\n",process_status);
        if(WIFEXITED(process_status) != 0)
        {
            printf("DETECTED NORMAL\n");
            if(WEXITSTATUS(process_status) != 0)
            {
                printf("DETECTED ERROR\n");

/*
 * If exited due to error, kill all other segments, report error, and die.
 */

                sprintf(tmp_err_str,
                    "FATL: Process %d running segment %d "
                    "terminated with error.",
                    current_pid,index);
                error_handler("par_bill.pc",UNKNOWN,tmp_err_str);
                par_per.status = -1;
                memcpy(shmaddress,&par_per,
                    sizeof(struct par_per_struct));
                seg_per.running = 0;
                shmaddress_s = (shmaddress +
                    (sizeof(struct par_per_struct) +
                     ((segment_list->segment_number - 1) *
                      sizeof(struct seg_per_struct))));
                memcpy(shmaddress_s,&seg_per,
                    sizeof(struct seg_per_struct));
                kill(0,SIGKILL);
            }/* _exit(1) */
        }/* process terminated normally */
        else if(WIFSIGNALED(process_status) != 0)
        {
            printf("DETECTED KILL\n");
            /* Report that process was killed and kill all */

```

```

/* others before exiting. */
sprintf(tmp_err_str,
        "FATL: Process %d running segment %d was killed "
        "by signal %d",
        current_pid, index, WTERMSIG(process_status));
error_handler("par_bill.pc", UNKNOWN, tmp_err_str);
par_per.status = -1;
memcpy(shmaddress, &par_per, sizeof(struct par_perf_struct));
seg_per.running = 0;
shmaddress_s = (shmaddress +
                (sizeof(struct par_perf_struct) +
                 ((segment_list->segment_number - 1) *
                  sizeof(struct seg_perf_struct))));
memcpy(shmaddress_s, &seg_per,
        sizeof(struct seg_perf_struct));
kill(0, SIGKILL);
}/* Killed by signal */
#ifdef _SEQUENT_
else if (WIFCORESIG(process_status) != 0)
#else
else if (WCOREDUMP(process_status) != 0)
#endif
{
    printf("DETECTED CORE\n");
    sprintf(tmp_err_str,
            "FATL: Process %d running segment %d was "
            "killed by signal %d causing core dump.",
            current_pid, index, WTERMSIG(process_status));
    error_handler("par_bill.pc", UNKNOWN, tmp_err_str);
    par_per.status = -1;
    memcpy(shmaddress, &par_per, sizeof(struct par_perf_struct));
    seg_per.running = 0;
    shmaddress_s = (shmaddress +
                    (sizeof(struct par_perf_struct) +
                     ((segment_list->segment_number - 1) *
                      sizeof(struct seg_perf_struct))));
    memcpy(shmaddress_s, &seg_per,
            sizeof(struct seg_perf_struct));
    kill(0, SIGKILL);
}/* Core dump */
else if (WSTOPSIG(process_status) != 0)
{
    printf("DETECTED STOP\n");
    sprintf(tmp_err_str,
            "FATL: Process %d running segment %d was "
            "stopped by signal %d.",
            current_pid, index, WTERMSIG(process_status));
    error_handler("par_bill.pc", UNKNOWN, tmp_err_str);
    par_per.status = -1;
    memcpy(shmaddress, &par_per, sizeof(struct par_perf_struct));
    seg_per.running = 0;
    shmaddress_s = (shmaddress +
                    (sizeof(struct par_perf_struct) +
                     ((segment_list->segment_number - 1) *
                      sizeof(struct seg_perf_struct))));
    memcpy(shmaddress_s, &seg_per,
            sizeof(struct seg_perf_struct));
    kill(0, SIGKILL);
}/* Stop signal */
wait_count = 0;
}
else
{
    if (current_pid == -1)
    {
        printf("process_status = %d\n", process_status);
    }
}

```

```

        sprintf(tmp_err_str,
            "WARN: monitor2: wait pid is finished. "
            "Parallel manager is terminating.");
        error_handler("par_bill.pc", UNKNOWN, tmp_err_str);
        finished = 1;
    }/* wait pid error dump */
    else
    {
        printf("process_status = %d\n", process_status);
        wait_count++;
        sprintf(tmp_err_str,
            "WARN: monitor2: No status was returned.");
        error_handler("par_bill.pc", UNKNOWN, tmp_err_str);

        sleep(5);
        if(wait_count == MAX_WAIT) finished = 1;
    }/* wait pid error dump */
    }/* Problems with wait pid */
}/* Monitor without creating replacements */
printf("FINISHED MONITOR 2.\n");

segment_list = segment_list_start;
while(segment_list != (struct segment_struct *)NULL)
{
    printf("%3.3s ", market);
    printf("%s ", segment_list->rpt_file);
    printf("%17.17s ", oracle_login);
    printf("%1.1s:", commit_flag);
    printf("%1.1s:", override_flag);
    printf("%1.1s ", dynamic_load);
    printf("%10.10s ", bill_date);
    printf("%10.10s ", segment_list->begin_acct);
    printf("%10.10s ", segment_list->end_acct);
    printf("%s ", segment_list->stdout_file);
    printf("%d:", segment_list->segment_number);
    printf("%d:", segment_list->process_id);
    printf("%d:", segment_list->processor);
    printf("%d:", segment_list->running);
    printf("%d ", segment_list->complete);
    printf("%ld ", segment_list->csize);
    printf("%ld\n", segment_list->asize);
    segment_list = segment_list->link;
}/* Show state of segment list when parallel manager terminated. */

}/* If not error logging into Oracle */
else
{
    error_handler("par_bill.pc", UNKNOWN, "Can't log in to ORACLE");
    error = TRUE;
    par_per.status = -1;
    memcpy(shmaddress, &par_per, sizeof(struct par_perf_struct));
}/* If oracle error logging in */

/* free segment list memory */
segment_struct *segment_tmp = segment_list = segment_list_start;
while (segment_list)
{
    segment_list = segment_list->link;
    free(segment_tmp);
    segment_tmp = segment_list;
}

if ((oracleLogin(oracle_login, NULL)) != -1)
{
    if ((!error) && (reports_flag) && (number_of_segments > 1))
    {

```

```

shmark_time(2,mark_time_arr,1);
par_per.status = 3;
memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));
error = prt_bill_rpts(market,bill_date,number_of_segments);
shmark_time(2,mark_time_arr,2);
memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));

/* Merge utility not installed */
shmark_time(3,mark_time_arr,1);
par_per.status = 4;
memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));
/* error = merge_bill_rpts() */
shmark_time(3,mark_time_arr,2);
memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));

}/* generate reports if selected */
}/* If not error logging into Oracle */
else
{
    error_handler("par_bill.pc",UNKNOWN,
        "Can't log in to ORACLE for reporting");
    error = TRUE;
}/* If oracle error logging in*/

if(error)
{
    error_handler("par_bill.pc",UNKNOWN,"prt_bill_rpts returned error");
    par_per.status = -1;
    memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));
}/* generate reports */
else
{
    par_per.status = 0;
    memcpy(shmaddress,&par_per,sizeof(struct par_perf_struct));
}

/* Don't need database anymore. */
oracleLogout();

shmark_time(0,mark_time_arr,2);

return 0;
}/* test main */

void fork_segment(segment_struct *segment,
    char arg_list[ARG_COUNT][MAX_ARG_SIZE],
    char *shmaddress,char *executable)
{
    char tmp_err_str[80];
    char *shmaddress_s;

    /* Set up segment specific arguments execution */
    sprintf(arg_list[2],"%s",segment->rpt_file);
    sprintf(arg_list[8],"%d",segment->segment_number);
    sprintf(arg_list[9],"%s",segment->begin_acct);
    sprintf(arg_list[10],"%s",segment->end_acct);
    sprintf(arg_list[11],"%s",segment->stdout_file);

```

```

/* flush before fork to avoid stdio file inconsistencies */
fflush(stdout);

if((segment->process_id = vfork()) == 0)
{
    /* Set stdout descriptor to close on successful exec only. */
    fcntl(1, F_SETFD, 1);
    /* Exec a bill segment */
    if(execl(executable, arg_list[0],
            arg_list[1],
            arg_list[2],
            arg_list[3],
            arg_list[4],
            arg_list[5],
            arg_list[6],
            arg_list[7],
            arg_list[8],
            arg_list[9],

```

```

        arg_list[10].
        arg_list[11].
        arg_list[12]) == -1)
    {
        sprintf(tmp_err_str,
            "FATL: Failed to exec segment %d", segment->segment_number);
        error_handler("par_bill.pc", UNKNOWN, tmp_err_str);

        par_per.status = -1;
        memcpy(shmaddress, &par_per, sizeof(struct par_per_struct));
        seg_per.running = 0;
        shmaddress_s = (shmaddress +
            (sizeof(struct par_per_struct) +
            ((segment->segment_number - 1) *
            sizeof(struct seg_per_struct))));
        memcpy(shmaddress_s, &seg_per,
            sizeof(struct seg_per_struct));
        /* Kill off process group first, then exit */
        kill(0, SIGKILL);
    }
}
else if(segment->process_id != 0)
{
    segment->running = 1;
    printf("process created = %d\n", segment->process_id);
} /* Parent should log segment as a running segment */
}

void shmark_time(int remark_nr, mark_struct *time_array, int mark_number)
{
    int error=0;
    int sequential=0;
    int tmp=0;
    time_t curtime;
    struct tm *loc_time;

    /* set the minutes west of Greenwich and timezone treatment */
    if (curtime = time(0))
    {
        loc_time = localtime(&curtime);
        /* determine the elapsed time since the last mark */
        if (mark_number == 1)
        {
            printf("%s %s", time_array[remark_nr].remark, asctime(loc_time));
        }
        if (mark_number == 2)
        {
            printf("%s - time elapsed since last mark: secs %f\n",
                time_array[remark_nr].remark,
                (float)((float)curtime -
                (float)time_array[remark_nr].seconds));
            if(remark_nr == 1)
            {
                par_per.load_bal_time =
                    curtime - time_array[remark_nr].seconds;
            }
            else if(remark_nr == 2)
            {
                par_per.rpt_build_time =
                    curtime - time_array[remark_nr].seconds;
            }
            else if(remark_nr == 3)
            {
                par_per.rpt_merge_time =
                    curtime - time_array[remark_nr].seconds;
            }
        }
    }
}

```



```
    )  
    time_array[remark_nr].seconds = curtime; /* ptx conversion */  
  )  
)
```

```

#define MAX_PROCS 50
#define MAX_WAIT 100
#define ARG_COUNT 13
#define MAX_ARG_SIZE 30
#define SHARED_MEM_KEY 100

```

```

#include <sys/types.h>
#include "par_man_proto.h"

```

```

struct segment_struct
{
    char      market[4];
    char      rpt_file[25];
    char      oracle_login[18];
    char      commit_flag[2];
    char      override_flag[2];
    char      bill_date[11];
    char      begin_acct[11];
    char      end_acct[11];
    char      stdout_file[25];
    long      csize;
    long      asize;
    long      row_num;
    long      count;
    int       segment_number;
#ifdef _SEQUENT_
    int       process_id;
#else
    pid_t     process_id;
#endif
    int       processor;
    int       running;
    int       complete;
    struct segment_struct *link;
};

```

```

struct acct_range
{
    char begin_acct[10];
    char end_acct[10];
    struct acct_range *link;
};

```

```

struct merge_struct
{
    int       segment_number;
    int       process_id;
    int       processor;
    int       running;
    int       complete;
    struct merge_struct *link;
};

```

```

struct seg_perf_struct
{
    int       seg_bills;
    int       seg_accts;
    int       segment_number;
#ifdef _SEQUENT_
    int       process_id;
#else
    pid_t     process_id;
#endif
    int       processor;
    int       running;
    int       complete;
};

```

```

long      slow_time;
long      fast_time;
long      last_acct_time;
long      last_cust_time;
long      elapsed_time;
long      total_time;
long      bill_count;
long      acct_count;
char      last_account[10];
char      last_cust[10];
};

```

```

struct par_perf_struct

```

```

{
    int      segments;
    int      status;
    long     load_bal_time;
    long     rpt_build_time;
    long     rpt_merge_time;
};

```

```

/* status values definition

```

```

= 0 - terminated normally

```

```

> 0 - status (1 - load; 2 - bill exec; 3 - report build; 4 - report merge)

```

```

< 0 - abnormal termination signal code

```

```

*/

```

```

#include <stdlib.h>
#ifdef _SEQUENT_
#include "parallel/parallel.h"
#include <sys/tmp_ctl.h>
#endif
#include "stddevlp.h"

int get_cpus()
{
    /* default cpus for a non-parallel machine */
    int cpu_count=1;

    /* Get number of CPUs */
#ifdef _SEQUENT_
    cpu_count = cpus_online();
#endif

    return (cpu_count);
}

```

```

/*****
* Name      : error_handler
*
* Description : The billing system error handling routine.
*
* Parameters : f_name - the function calling the error routine.
*              error_code - error message code.
*              info - additional error information.
*
* Return Value : void.
*
*
*
*
*
*
* Notes
*****/

#include <stdio.h>
#include <string.h>
#include <time.h>
#include "bglobal.h"
#include "vg_error.h"

void error_handler(char *f_name,int error_code,char *info)
/* char *f_name - function name */
/* int error_code - error code */
/* char *info - additional information e.g. filename of open file */
{
    FILE      *fp; /* file pointer to error log file */
    char      message[ERR_MESSAGE_LENGTH+1];
    char      *err_log_fn = "vgerr.log";
    time_t    curtime; /* current time in seconds */

    /* print any additional instructions and set the return status */
    switch (error_code)
    {
        case QTEL_DB:
            strcpy(message,"error updating QTEL database");
            break;
        case TAPE_READ:
            strcpy(message,"error reading tape");
            break;
        case FILEOPEN:
            sprintf(message,"can't open file %s",info);
            break;
        case FILECLOSE:
            sprintf(message,"can't close file %s",info);
            break;
        case FWRITE:
            sprintf(message,"fwrite error in file %s",info);
            break;
        case FREAD:
            sprintf(message,"fread error in file %s",info);
            break;
        case FSEEK:
            sprintf(message,"fseek error in file %s",info);
            break;
        case ORACLELOG:
            strcpy(message,"can't log on to oracle");
            break;
        case ORACLECREATE:
            sprintf(message,"can't create the table %s",info);
            break;
        case ORACLEINSERT:
    }
}

```

```

    sprintf(message,"can't insert %s",info);
    break;
case ORACLEDELETE:
    sprintf(message,"can't insert %s",info);
    break;
case ORACLESELECT:
    sprintf(message,"can't select %s",info);
    break;
case ORACLEUPDATE:
    sprintf(message,"can't update %s",info);
    break;
case ORACLENOTFOUND:
    sprintf(message,"table not found %s",info);
    break;
case SYS_ERROR:
    sprintf(message,"cannot execute the system call %s",info);
    break;
default:
    sprintf(message,"UNKNOWN error %s",info);

} /* switch error_code */

/* write the error message to the error log file */

/* if the log file does not exist then create it */
/* NOTE: The use of "a+" to append and/or create to append is not in */
/* accordance with the ansi standard and may cause upgrade and/or port */
/* problems. */
if ( (fp = fopen(err_log_fn,"a+")) != NULL)
{
    if ((curtime = time(0)) != -1)
    {
        fprintf(fp,"%s error in %s : %s\n",ctime(&curtime),
                f_name,message);
    } /* if time of day */
    else
    {
        printf("\nCan't get the time of day value\n");
    } /* else error */

    if (fclose(fp))
    {
        printf("\nError handler: can't close the error log file\n");
        printf("%s error in %s : %s\n",ctime(&curtime),
                f_name,message);
    } /* if fclose */
} /* append to existing or open new log file */
else
{
    printf("\nError handler: can't open the error log file\n");
    printf("%s error in %s : %s\n",ctime(&curtime),
            f_name,message);
} /* can't open error log file */

} /* error_handler */

```

```

#ifndef __PAR_MAN_PROTO_H
#define __PAR_MAN_PROTO_H

int get_distribution(struct segment_struct **segment_list,
                    char *market,
                    long number_of_segments,
                    char *dynamic_load,
                    char *start_account,
                    char *end_account);

int get_cpus();

void error_handler(char *f_name, int error_code, char *info);
BOOLEAN prt_bill_rpts(char *mkt, char *billdate, long segment_count);
BOOLEAN get_executable(char *path, char *name);

#endif /* __PAR_MAN_PROTO_H */

```

```

#define PROJECT_MAIN
#define BILL_TEST
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include "bill_global.h"
#include "bill_struct.h"
#include "comments.h"
#include "stddevlp.h"
#include "vg_error.h"
#include "error.h" /* REV1 */
#include "error_proto.h"
#ifdef _SEQUENT_
#include <sys/tmpctl.h>
#endif
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <time.h>
#include "taxlib.h"
#include "bill_proto.h"
#include "bparallel.h"

char *a;

#ifdef _SEQUENT_
extern "C" char *sbrk(int);
#endif

struct ora_tab_struct
{
    char table_name[81];
    long seconds;
    long useconds;
};

/* These are global for diagnostic development purposes. */
int segment=0;
struct ora_tab_struct oracle_tables[10];

#pragma sequent_expandable (printf(), fprintf(), memcpy(), fwrite())
EXEC SQL BEGIN DECLARE SECTION;
    static VARCHAR    uid[80]; /* user id */
    static char        omarket[3]; /* bill date validation kludge */
    static char        obill_date[8]; /* bill date validation kludge */
    static VARCHAR    obill_date2[10]; /* thp - bill date validation */
    static VARCHAR    obill_date_test[10]; /* thp - bill date validation */
EXEC SQL END DECLARE SECTION;
#undef SQLCA_STORAGE_CLASS
EXEC SQL INCLUDE SQLCA.H;
EXEC ORACLE OPTION (MAXOPENCURSORS=30);

struct mark_struct
{
    char remark[81];
    long seconds;
    long useconds;
};

void mark_time(int remark_nr, mark_struct *time_array, int mark_number);

GLOBAL TaxInterface *taxer;

```


[illegible]

```

struct recur_struct *recur_list = (struct recur_struct *)NULL;
/* customer recurring charges */
struct recur_struct *misc_mkt_chg = (struct recur_struct *)NULL;
struct non_recur_struct *nonrecur_list = (struct non_recur_struct *)NULL;
/* customer nonrecurring charges */
struct call_struct *call_list = (struct call_struct *)NULL; /* call list */
struct cust_struct *cust_info_list = (struct cust_struct *)NULL;
/* customer information list */
struct tod_desc_struct *tod_desc_list = (struct tod_desc_struct *)NULL;
/* tod description list */

struct bill_info_struct bill_info_rec; /* billing information record */
memset(&bill_info_rec, NULL, sizeof(bill_info_struct));

struct exemption_info *exemption_list = (exemption_info *)NULL;

struct ar_struct *ar_list = (struct ar_struct *)NULL; /* A/R information list */
struct collect_adj_struct *collect_adj_list = (struct collect_adj_struct *)NULL;
/* adjustments list for collections */
struct adjustment_struct *adjustment_list = (struct adjustment_struct *)NULL;
/* adjustments list */
struct fyi_notice_struct *fyi_messages = (struct fyi_notice_struct *)NULL;
/* for your information list */

struct date_struct todays_date; /* todays_date */
struct date_struct latefee_date; /* date of latefee threshold */
struct date_struct bill_date; /* bill cutoff date */
struct date_struct period_date; /* billing period start or end date */
struct date_struct due_date; /* bill due date */
struct date_struct prorate_to_date; /* prorate to date */
struct date_struct prorate_from_date; /* prorate from date */
struct date_struct activation_date; /* customer activation date */
struct date_struct deactivation_date; /* customer deactivation date */
struct date_struct suspend_date; /* customer suspend date */
struct date_struct offset_display_date; /* bill date + offset */
int i; /* loop control and indexing */
struct airtime_summary_struct *airtime_summary =
    (struct airtime_summary_struct *)NULL;
/* airtime summary for reporting */
struct report_format rev_rpt_struct; /* account receivable report structure */
struct report_format ar_rpt_struct; /* account receivable report structure */
char **as_rpt; /* pointer to airtime summary report */
struct report_format as_rpt_struct; /* airtime summary report structure */
char **tas_rpt; /* pointer to toll and airtime summary report */
struct report_format tas_rpt_struct; /* toll and airtime summary report struct */
struct toll_airtime_struct *toll_airtime_list =
    (struct toll_airtime_struct *)NULL;
/* toll and airtime summary for reporting */
struct totals_struct total_non_call_totals; /* non call totals for market */
memset(&total_non_call_totals, NULL, sizeof(totals_struct));
struct call_totals_struct total_call_totals; /* call totals for market */
memset(&total_call_totals, NULL, sizeof(call_totals_struct));
struct call_totals_struct total_roamer_totals; /* roamer totals for */
/* market */
memset(&total_roamer_totals, NULL, sizeof(call_totals_struct));
char **billing_rpt; /* pointer to billing report */
struct report_format billing_rpt_struct; /* billing report struct */
char **js_rpt; /* pointer to journal summary report */
struct report_format js_rpt_struct; /* journal summary report struct */
struct journal_struct *journal_list = (struct journal_struct *)NULL;
/* journal summary for reporting */
char **ps_rpt; /* pointer to phone sales report */
struct report_format ps_rpt_struct; /* phone sales report struct */

```

```

struct tax_reg_summary *tax_register = (tax_reg_summary *)NULL;
/* tax register by geocode */

struct report_format zero_rpt_struct; /* zero bill report struct*/
struct report_format sxcp_rpt_struct; /* exception report struct*/
struct report_format dxcp_rpt_struct; /* exception report struct*/
char **tr_rpt; /* pointer to tax register report */
struct report_format tr_rpt_struct; /* tax register report struct*/
char **chrg_rpt; /* pointer to charge detail report */
struct report_format chrg_rpt_struct; /* charge detail report struct*/
char **comw_rpt; /* pointer to commission waivers report */
struct report_format comw_rpt_struct; /* commission waivers report struct*/
struct phone_sales_list_struct *phone_sales_list_header =
    (phone_sales_list_struct *)NULL; /* charge type header */
struct phone_sales_list_struct *phone_sales_list_header_cur =
    (phone_sales_list_struct *)NULL; /* charge type current */
struct phone_sales_tot_struct *phone_sales_list =
    (struct phone_sales_tot_struct *)NULL;
/* phone sales for reporting */

struct cur_charge_struct *cur_charge_list =
    (struct cur_charge_struct *)NULL;
/* charge list start */

BOOLEAN bill_commit = FALSE; /* TRUE if this run is a commit billing */
BOOLEAN override = FALSE; /* TRUE if no abort on date errors */
char *temp_list_start; /* generic pointer used to free linked lists */
struct bill_format bp; /* bill page format structure */
struct bill_format dbp; /* detail bill page format structure */
struct cust_struct *master_aggregate_ptr; /* master aggregate pointer */
/* while processing an aggregate account */
struct aggregate_struct *aggregate_totals = (struct aggregate_struct *)NULL;
/* list of aggregate totals */
struct aggregate_struct *aggregate_totals_start =
    (struct aggregate_struct *)NULL; /* list of aggregate totals */
BOOLEAN processing_aggregate = FALSE; /* TRUE if currently processing an */
/* aggregate account */
struct p_category_struct *cat_list = (struct p_category_struct *)NULL;
/* adjustment print category list */
char prev_acct_nr[10]; /* previous account number being processed */
int airtime_detail_start; /* starting page of airtime detail */

struct commwaiv_struct *comw_list = (struct commwaiv_struct *)NULL;

long comw_amt_totals = 0L;
long comw_fed_totals = 0L;
long comw_state_totals = 0L;
long comw_county_totals = 0L;
long comw_loc_totals = 0L;

struct mark_struct mark_time_arr[20];
struct collections_info dunning_cust; /* Node of customer information for
late notice */
memset(&dunning_cust, NULL, sizeof(collections_info));
struct zero_bill_struct *zero_bill_list = (zero_bill_struct *)NULL;
/* pointer of customer information for
zero bill report */
struct collections_stat_hdr dunning_stats_hdr;
memset(&dunning_stats_hdr, NULL, sizeof(collections_stat_hdr));
struct collections_stat
    *dunning_stats = (struct collections_stat *)NULL;
struct collections_info *dunning_exception_list =

```

```

(struct collections_info *)NULL; /* List of dunning exceptions */
BOOLEAN send_bill=FALSE;
struct duedate_list *ddl_list = (struct duedate_list *)NULL; /* due_date list */
struct free_number_struct *free_number_ptr; /* freenumber table (roam america) */

call_struct *taxable_calls = (call_struct *)NULL;

struct super_list *super = (struct super_list *)NULL;
struct write_off *temp_write_off = (struct write_off *)NULL;
struct debt_exception *temp_debt_xcp = (struct debt_exception *)NULL;
struct journal_ref *temp_jour_ref = (struct journal_ref *)NULL;
struct rev_total *temp_rev_total = (struct rev_total *)NULL;
struct bill_parameter *temp_bill_params = (struct bill_parameter *)NULL;

/* ----- */
/* - Call discounting variables and functions */
/* ----- */
struct discountPlan plan;
char pfile_buf[155648 * 2];
char pfile_buf_tmp[155648];
char bfile_buf[155648 * 2];
char bfile_buf_tmp[155648];

char sxcp_file[30];
char dxcp_file[30];
char zero_file[30];
char ar_rpt_file[30];
char as_rpt_file[30];
char tas_rpt_file[30];
char js_rpt_file[30];
char ps_rpt_file[30];
char tr_rpt_file[30];
char comw_rpt_file[30];
char rev_chg_rpt_file[30];
char billing_rpt_file[30];

BOOLEAN reopen_flag=FALSE;
BOOLEAN parallel=FALSE;
char diag_file_name[40];
char diag2_file_name[40];
char error_filename[40];
/* ----- */
/* - Call discounting variables and functions */
/* ----- */
FILE *fpstd;
FILE *fpstde;

/* Shared memory interface variables */
key_t shbill_key=SHARED_MEM_KEY;
key_t shbill_id=0;
char *shmaddress; /* Pointer to shared memory */
char tmp_err_buf[80]; /* for more descriptive error messages */

strcpy(mark_time_arr[0].remark, "BDRPT - NEW CUSTOMER");
mark_time_arr[0].useconds = 0L;
mark_time_arr[0].seconds = 0L;
strcpy(mark_time_arr[1].remark, "MTIME - POST PAYMENTS");
mark_time_arr[1].useconds = 0L;
mark_time_arr[1].seconds = 0L;
strcpy(mark_time_arr[2].remark, "MTIME - POST CALLS (HOME)");
mark_time_arr[2].useconds = 0L;
mark_time_arr[2].seconds = 0L;
strcpy(mark_time_arr[3].remark, "MTIME - RATE LOCAL HOME AIRTIME");

```

```

mark_time_arr[3].useconds = 0L;
mark_time_arr[3].seconds = 0L;
strcpy(mark_time_arr[4].remark,"MTIME - PRINT BILL");
mark_time_arr[4].useconds = 0L;
mark_time_arr[4].seconds = 0L;
strcpy(mark_time_arr[5].remark,"MTIME - TOTAL BILL PROCESS");
mark_time_arr[5].useconds = 0L;
mark_time_arr[5].seconds = 0L;
strcpy(mark_time_arr[6].remark,"MTIME - RPT DATA INSERT");
mark_time_arr[6].useconds = 0L;
mark_time_arr[6].seconds = 0L;
strcpy(mark_time_arr[7].remark,"MTIME - POST CALLS (ROAM)");
mark_time_arr[7].useconds = 0L;
mark_time_arr[7].seconds = 0L;
strcpy(mark_time_arr[8].remark,"MTIME - CALC ROAM (ROAM)");
mark_time_arr[8].useconds = 0L;
mark_time_arr[8].seconds = 0L;
strcpy(mark_time_arr[9].remark,"SUMMARY USAGE 2");
mark_time_arr[9].useconds = 0L;
mark_time_arr[9].seconds = 0L;
strcpy(mark_time_arr[10].remark,"SUMMARY USAGE 3");
mark_time_arr[10].useconds = 0L;
mark_time_arr[10].seconds = 0L;
strcpy(mark_time_arr[11].remark,"SUMMARY USAGE 4");
mark_time_arr[11].useconds = 0L;
mark_time_arr[11].seconds = 0L;
strcpy(mark_time_arr[12].remark,"SUMMARY USAGE 5");
mark_time_arr[12].useconds = 0L;
mark_time_arr[12].seconds = 0L;
strcpy(mark_time_arr[13].remark,"MTIME - RPT DATA INSERT");
mark_time_arr[13].useconds = 0L;
mark_time_arr[13].seconds = 0L;

```

```

// clear out plan struct
memset(&plan, NULL, sizeof(discountPlan));

```

```

// set up error handler information
setIdentity(argv[0]);
setErrorFile("vgerr.log");

```

```

/* Set I/O buffer size for standard out
setvbuf(stdout, (char) NULL, _IOFBF, 65536); */

```

```

mark_time(5,mark_time_arr,1);

```

```

strcpy(market,argv[1]);
if (argv[4] != (char) NULL)
{
    sscanf(argv[4],"%2d/%2d/%4d",&bill_date.month,&bill_date.day,
        &bill_date.year);
    sprintf(bill_date.date_str,"%4d%02d%02d",bill_date.year,
        bill_date.month,bill_date.day);
} /* if arg passed */
else
{
    bill_date.year = 0;
    bill_date.month = 0;
    bill_date.day = 0;
} /* else no arg passed */

```

```

memcpy(obill_date,bill_date.date_str,8);
memcpy(omarket,market,3);

```

```

vput(&obill_date2, argv[4]);

```

```

/* ----- */
/* - Set the error log for the changes that use the */
/* usererr function for reporting error from billing. */
/* ----- */
open_error_log("vgerr.log");

if (*argv[5] == '1')
    bill_commit = TRUE;
if (*argv[6] == '1')
    override = TRUE;
if (*argv[7] == 'P')
    parallel = TRUE;

if ((segment = ((int)atoi(argv[8]))) == 0)
{
    error_handler("bill_test.pc", UNKNOWN,
        "Could not determine segment number.");
    _exit(1);
}

if (parallel)
    sprintf(ar_rpt_file, "ar_%d.rpt", segment);
else
    sprintf(ar_rpt_file, "ar.rpt");

sprintf(as_rpt_file, "as.rpt");
sprintf(tas_rpt_file, "tas.rpt");
sprintf(js_rpt_file, "js.rpt");
sprintf(ps_rpt_file, "ps.rpt");
sprintf(tr_rpt_file, "tr.rpt");
sprintf(comw_rpt_file, "comw.rpt");
sprintf(rev_chg_rpt_file, "rev_chg.rpt");
sprintf(billing_rpt_file, "billing.rpt");
sprintf(diag_file_name, "%s.xxx", argv[11]);
sprintf(diag2_file_name, "%s.err", argv[11]);

if((fpstd = freopen(diag_file_name, "w", stdout)) == (FILE *)NULL)
{
    error_handler("bill_test.pc", FILEOPEN,
        "Could bill diagnostic file");
    _exit(1);
}/* Can't open diagnostic file */
else
{
    if((fpstd = freopen(diag2_file_name, "w", stderr)) == (FILE *)NULL)
    {
        error_handler("bill_test.pc", FILEOPEN,
            "Couldn't open stderr bill diagnostic file");
        _exit(1);
    }/* Can't open diagnostic file */

    sprintf(tmp_err_buf, "sbrk: %d", sbrk(0));
    error_handler("par_bill.pc", UNKNOWN, tmp_err_buf);
#ifdef _SEQUENT_
    shbill_id = shmget(shbill_key, 0, IPC_CREAT);
#else
    shbill_id = shmget((int)shbill_key, 0, IPC_CREAT);
#endif
    sprintf(tmp_err_buf, "sbrk: %d", sbrk(0));
    error_handler("par_bill.pc", UNKNOWN, tmp_err_buf);

    if(shbill_id == -1)
    {
        error = 1;
        sprintf(tmp_err_buf,
            "Shared memory allocation for %d: attempt failed.", shbill_key);
    }
}

```

```

        error_handler("bill_test.pc",UNKNOWN,tmp_err_buf);
        _exit(0);
    }/* Get new key if in use */
    else
    {
/* Attach shared memory segment */
// #ifdef _SEQUENT_
// shmaddress = shmat(shbill_id,0,0);
// #else
shmaddress = (char *)shmat((int)shbill_id,(void *)0,0);
// #endif
sprintf(tmp_err_buf,"sbrk: %d",sbrk(0));
error_handler("par_bill.pc",UNKNOWN,tmp_err_buf);
if(((int )shmaddress) == -1)
{
    sprintf(tmp_err_buf,"Chimp3 %d",errno);
    perror(tmp_err_buf);
    error = TRUE;
    sprintf(tmp_err_buf,
        "Could not attach shared memory in segment %d.",segment);
    error_handler("bill_test.pc",UNKNOWN,tmp_err_buf);
    _exit(1);
}
else
{
/* Set shared memory address to that of this segments shared area */
shmaddress += (sizeof(struct par_perf_struct) +
                ((segment-1) *
                 sizeof(struct seg_perf_struct)
                ));

memcpy(&seg_perf,shmaddress,sizeof(struct seg_perf_struct));
seg_perf.segment_number = segment;
seg_perf.running = 1;
seg_perf.complete = 0;
seg_perf.slow_time = 0;
seg_perf.fast_time = 100;
seg_perf.last_acct_time = 0;
seg_perf.last_cust_time = 0;
seg_perf.elapsed_time = 0;
seg_perf.total_time = 0;
seg_perf.bill_count = 0;
seg_perf.acct_count = 0;
memcpy(seg_perf.last_account,
        "XXXXXXXXXX",10);
memcpy(seg_perf.last_cust,
        "XXXXXXXXXX ",10);

/* Initialize shared memory for this treagment. */
memcpy(shmaddress,&seg_perf,(sizeof(struct seg_perf_struct)));

sprintf(tmp_err_buf,"sbrk: %d",sbrk(0));
error_handler("par_bill.pc",UNKNOWN,tmp_err_buf);
}
} /* got shmget() */

setvbuf(stdout,(char)NULL,_IOFBF,65536);
printf("ts ts ts ts ts ts ts ts ts ts ts\n",
    argv[0],
    argv[1],
    argv[2],
    argv[3],
    argv[4],
    argv[5],
    argv[6],
    argv[7],

```

```

        argv[8],
        argv[9],
        argv[10],
        argv[11],
        argv[12]);
    }/* TESTING REMOVE */

/* log on to oracle */
strcpy((char *)uid.arr,argv[3]);
uid.len = strlen((char *)uid.arr);
EXEC SQL CONNECT :uid;
if (sqlca.sqlcode == NOT_SQL_ERROR)
{
/*
EXEC SQL ALTER SESSION SET OPTIMIZER_GOAL = RULE;
EXEC SQL ALTER SESSION SET SQL_TRACE TRUE;
*/
/*
EXEC SQL SELECT TO_CHAR(TO_DATE(:obill_date2, 'mm/dd/YYYY')) INTO :obill_date_test FROM DUAL;

if (sqlca.sqlcode != 0)
{
    error_handler("bill_test.pc",UNKNOWN,
    "FATAL ERROR : bill date parameter is not in mm/dd/YYYY format.");
    exit(0);
}/* If error, abort and inform operator to check bill date */
/* thp - end new kludge */

/* HUGE VANGUARD KLUDGE FOR bill date validation */
EXEC SQL SELECT BILL_DATE INTO :obill_date2 FROM SWITCH_MARKET WHERE
MARKET = :omarket AND
BILL_DATE = ADD_MONTHS(TO_DATE(:obill_date,'YYYYMMDD'),-1);

if ((sqlca.sqlcode != 0))
{
    error_handler("bill_test.pc",UNKNOWN,
    "FATAL ERROR : bill date parameter is not 1 month greater than last bill date.");
    _exit(0);
}/* If error, abort and inform operator to check bill date */

// wholt 12/6/92 changed for new tax lib
taxer = new TaxInterface;
*/
sprintf(print_fn, "/dev/null");
sprintf(print_tmp_fn, "%s.prt.tmp", argv[2]);
*/
sprintf(print_fn, "%s.prt", argv[2]);
sprintf(print_tmp_fn, "%s.prt.tmp", argv[2]);
sprintf(bill_image_fn, "%s.bmg", argv[2]);
sprintf(bill_image_tmp_fn, "%s.bmg.tmp", argv[2]);

/*-----*/
/* Get the super_list from the database (rgates) */
/* -----*/
if (!bld_writeoff_list(&temp_write_off))
{
    add_sub_list(&super, temp_write_off, WRITEOFF);
}
if (!bld_debt_xcp_list(&temp_debt_xcp))
{
    add_sub_list(&super, temp_debt_xcp, DEBT_EXCEPT);
}
if (!bld_jrnl_ref_list(&temp_jour_ref))
{
    add_sub_list(&super, temp_jour_ref, JOURNAL_REFERENCE);
}

```



```

if(!bld_rev_total_list(&temp_rev_total))
{
    add_sub_list(&super,temp_rev_total,REVENUE_TOTAL);
}
if(!get_bill_params(&temp_bill_params,market))
{
    add_sub_list(&super,temp_bill_params,BILLING_PARAMS);
}

/* ----- */
/* - Get the discount plans from the database */
/* ----- */
if(retrieveDiscountPlans(&plan,market,bill_date.date_str) == -1)
{
    error_handler("Call Discounting",
        UNKNOWN,"Could not get discount plans");
    _exit(1);
}

/* name file by market */
if (((pfp = fopen(print_fn,"w+")) != NULL) &&
    ((bdfp = fopen(bill_image_fn,"w+")) != NULL))
{
    if(setvbuf(pfp,pfile_buf,_IOFBF,153600) == 0)
        if(setvbuf(bdfp,bfile_buf,_IOFBF,153600) == 0)

        /* build the free number list */
        free_number_ptr = get_free_list();
    /* retrieve the market information record */
    if (!get_market(market,&market_rec))
    {
        if (!get_due_list(market,&ddl_list))
        {
            if(!get_dunning_leeway(&market_rec.leeway_amount,
                &market_rec.latefee_leeway,
                market))
            {
                printf("notice %ld latefee %ld leeways\n",market_rec.leeway_amount,
                    market_rec.latefee_leeway);

                if (!get_rate_list(&rate_plan_list,market,
                    &airtime_summary))
                {
                    due_date.day = market_rec.due_date_day_in_month;
                    if (!get_date_values(&bill_date,&period_date,&due_date,&today's_date,
                        &latefee_date,(int)market_rec.latefee_threshold,
                        market_rec.init_pay_type,override,super))
                    {
                        if(strcmp(market_rec.bill_date.date_str,bill_date.date_str) == 0)
                        {
                            printf("FATAL ERROR: Current billing date is equal to last billing date.\n");
                            error_handler("bill_test.pc",UNKNOWN,
                                "Current bill_date = last bill date in switch_market table.");
                            _exit(0);
                        }
                    }
                    compute_billdate_offsets(&bill_date,&offset_display_date);
                    if ((tod_desc_list = get_tod_desc_list(market)) !=
                        (struct tod_desc_struct *)NULL)
                    {
                        misc_mkt_chg = get_misc_mkt_chg(market,&today's_date);
                        fyi_messages = get_fyi_notices(market,
                            &due_date,
                            &offset_display_date,
                            &market_rec.csh_rcvd_date,

```

```

                                &today's_date);
if(fyi_messages == (struct fyi_notice_struct *)NULL)
{
printf("FATAL ERROR: retrieving fyi message/late notices.\n");
error_handler("bill_test.pc",UNKNOWN,
"get_fyi_notices() returned fatal error.");
_exit(0);
}/* If fyi error fatal */

if ((cat_list = get_print_cat()) !=
(struct p_category_struct *)NULL)
{

printf("Going to get cust_list \n");
fflush(stdout);
if ((cust_info_list = get_cust_list(market,&bill_date,
argv[9],argv[10])) !=
(struct cust_struct *)NULL)
{
get_journal_summary(&journal_list);
get_phone_sales(&phone_sales_list,market);
get_phone_sales(&phone_sales_list,market,
temp_bill_params->ph_sales_jrnl_acct);

if ((phone_sales_list_header = (phone_sales_list_struct *)
malloc(sizeof(phone_sales_list_struct))) !=
(phone_sales_list_struct *) NULL)
{
phone_sales_list_header->sales_list = phone_sales_list;
strcpy(phone_sales_list_header->titleText, "PHONE");
phone_sales_list_header_cur = phone_sales_list_header;
phone_sales_list_header_cur->link = (phone_sales_list_struct *)NULL;

/* get 'RE' codes list */
if ((phone_sales_list_header_cur->link =
(phone_sales_list_struct *)malloc(sizeof(phone_sales_list_struct)))
!= (phone_sales_list_struct *) NULL)
{
phone_sales_list_header_cur = phone_sales_list_header_cur->link;
strcpy(phone_sales_list_header_cur->titleText, "EQUIPMENT");
phone_sales_list_header_cur->link = (phone_sales_list_struct *)NULL;
phone_sales_list_header_cur->sales_list =
(phone_sales_tot_struct *)NULL;
get_phone_sales(&(phone_sales_list_header_cur->sales_list),market, temp_bill_params->equip_sales_jrnl_acct);
}
else
{
error_handler("bill_test.pc",UNKNOWN,
"Malloc error for phone_sales_list_header.");
printf("ERROR OCCURRED BUILDING PHONE SALES LIST.\n");
}
}
else
{
error_handler("bill_test.pc",UNKNOWN,
"Malloc error for phone_sales_list_header.");
printf("ERROR OCCURRED BUILDING PHONE SALES LIST.\n");
}

if((get_rev_list(&rev_list,market)) != 0)
{
error_handler("bill_test.pc",UNKNOWN,
"Can't make revenue by charge code list. ");
printf("ERROR OCCURRED BUILDING REVENUE LIST.\n");
}
}

```

```

/*
    traverse(&rev_list);
*/

/* set the prorating to date as bill date */
prorate_to_date = bill_date;

/* initialize the report structures */
init_bill_rpt(&ar_rpt_struct, &as_rpt_struct, &tas_rpt_struct,
             &billing_rpt_struct, &js_rpt_struct,
             &ps_rpt_struct, &tr_rpt_struct,
             &chrg_rpt_struct, &comw_rpt_struct, &bill_date,
             &market_rec.super);

/* open the report files only in sequential mode */
if((parallel) || (!parallel) && ((
    (as_rpt_struct.rpt_file =
        fopen(as_rpt_file, "w+")) != NULL))
    && (
        (tas_rpt_struct.rpt_file =
            fopen(tas_rpt_file, "w+")) != NULL))
    && (
        (js_rpt_struct.rpt_file =
            fopen(js_rpt_file, "w+")) != NULL))
    && (
        (ps_rpt_struct.rpt_file =
            fopen(ps_rpt_file, "w+")) != NULL))
    && (
        (tr_rpt_struct.rpt_file =
            fopen(tr_rpt_file, "w+")) != NULL))
    && (
        (rev_rpt_struct.rpt_file =
            fopen(rev_chg_rpt_file, "w+")) != NULL))
    && (
        (billing_rpt_struct.rpt_file =
            fopen(billing_rpt_file, "w+")) != NULL))
    && (
        (comw_rpt_struct.rpt_file =
            fopen(comw_rpt_file, "w+")) != NULL))))
{
    /* open the ar report file IR regardless of parallel status */
    if(((ar_rpt_struct.rpt_file =
        fopen(ar_rpt_file, "w+")) == NULL))
    /*
        || (
        *   ((comw_rpt_struct.rpt_file =
        *       fopen(comw_rpt_file, "w+")) == NULL)))
    */
    {
        error_handler("bill_test", FILEOPEN,
            "ar report files");
        error = TRUE;
    } /* else fopen report files error */

    /* Set I/O buffer size for ar.rpt file */
    setvbuf(ar_rpt_struct.rpt_file, (char) NULL, _IOFBF, 102400);
    setvbuf(comw_rpt_struct.rpt_file, (char) NULL, _IOFBF, 102400);

    /* create the toll and airtime list for the home market */
    /* integrate into build market call list */
    if (!build_toll_airtime_list(&toll_airtime_list,
                                market_rec.market_sid,
                                market_rec.market_name))

```

```

{
    init_noncall_totals(&total_non_call_totals);
    init_call_totals(&total_call_totals);
    init_call_totals(&total_roamer_totals);
    init_dunning_stats(&dunning_stats_hdr,&dunning_stats);

    while (!error &&
           cust_info_list != (struct cust_struct *)NULL)
    {
        seg_perf.acct_count++;
        memcpy(seg_perf.last_account,
               cust_info_list->acct_nr,
               sizeof(cust_info_list->acct_nr));

mark_time(0,mark_time_arr,1);
        /* get the associated bill info record */
        if (!get_bill_info(&bill_info_rec,
                           cust_info_list->acct_nr))
        {
            /* get the current charges record */
            if (!get_current_charges(&cur_charge_list,
                                     cust_info_list->acct_nr,
                                     &bill_info_rec))
            {

                processing_aggregate = FALSE;
                do
                {
                    seg_perf.bill_count++;
                    memcpy(seg_perf.last_cust,
                           cust_info_list->cust_nr,
                           sizeof(cust_info_list->cust_nr));

printf("CUSTOMER # %-10.10s ACCT # %-10.10s\n",cust_info_list->cust_nr,
      cust_info_list->acct_nr);

                    memcpy(bill_info_rec.bill_categories,"00000000",8);
                    taxer->freeTaxList(&totals.noncall_tax);
                    taxer->freeTaxList(&totals.payment_adj_tax);
                    taxer->freeTaxList(&totals.home_adj_tax);
                    taxer->freeTaxList(&totals.foreign_adj_tax);
                    taxer->freeTaxList(&totals.payment_taxes);
                    taxer->freeTaxList(&totals.home_taxes);
                    taxer->freeTaxList(&totals.foreign_taxes);
                    taxer->freeTaxList(&current_charge_totals.noncall_tax);
                    taxer->freeTaxList(&current_charge_totals.payment_adj_tax);
                    taxer->freeTaxList(&current_charge_totals.home_adj_tax);
                    taxer->freeTaxList(&current_charge_totals.foreign_adj_tax);
                    taxer->freeTaxList(&current_charge_totals.payment_taxes);
                    taxer->freeTaxList(&current_charge_totals.home_taxes);
                    taxer->freeTaxList(&current_charge_totals.foreign_taxes);
                    init_noncall_totals(&totals);
                    init_noncall_totals(&current_charge_totals);

                    init_tax_rec(&totals.noncall_tax);
                    if(totals.noncall_tax != (struct vtax *)NULL)
                        taxer->freeTaxList(&totals.noncall_tax);

```

```

/* load date display variables */
load_date(&prorate_from_date,
          cust_info_list->activation_date);
load_date(&activation_date,
          cust_info_list->activation_date);
load_date(&deactivation_date,
          cust_info_list->deactivation_date);
load_date(&suspend_date, cust_info_list->suspend_date);
/* build the call related totals list */
if ((market_call_list =
     build_market_call_list(&market_rec)) !=
    (struct market_call_struct *)NULL)
{
/* if the customer element is a master aggregate */
/* reserve it to process after individual accounts */
if (cust_info_list->aggr == AGGREGATE_MASTER)
{
/* the first time through set up aggregates */
if (processing_aggregate == FALSE)
{
master_aggregate_ptr = cust_info_list;
/* point to the first sub account */
processing_aggregate = TRUE;

/* build the aggregate totals list */
build_aggr_totals_list(&aggregate_totals,
                      master_aggregate_ptr->cust_nr,
                      cust_info_list);

/* retrieve calls for each aggregate account */
mark_time(2, mark_time_arr, 1);
ret_aggr_call_info(aggregate_totals->link,
                  cust_info_list->link,
                  market_rec.market_sid,
                  &bill_date,
                  &(bill_info_rec.detail_sort_cd));
mark_time(2, mark_time_arr, 2);

calculate_free_aggr_airtime(aggregate_totals,
                          cust_info_list,
                          &bill_info_rec,
                          rate_plan_list,
                          &prorate_to_date,
                          &market_rec.bill_date,
                          &period_date,
                          market_rec.init_pay_type, &plan);

/* point to the first aggregate, if one exists. */
if (aggregate_totals->link !=
    (struct aggregate_struct *)NULL)
    aggregate_totals_start = aggregate_totals->link;
else
    aggregate_totals_start = aggregate_totals;

/* get the data for billing the first */
/* subordinate from the aggregate list */
market_call_list->call_list =
    aggregate_totals_start->call_list;

market_call_list->alt_call_list =
    aggregate_totals_start->alt_call_list;

/* copy the aggregate rate plan to current rate */
/* plan record */
copy_rate_plan(
    &aggregate_totals_start->rate_plan_rec,

```

```

        &customer_rate_plan);

/* if this aggregate master has no subordinates */
/* set processing aggregate flag to FALSE and */
/* process only the aggregate master */
if (memcmp(cust_info_list->acct_nr,
           cust_info_list->link->acct_nr,10))
{
    cust_info_list = master_aggregate_ptr;
    processing_aggregate = FALSE;
} /* if only master aggregate */
else
    cust_info_list = cust_info_list->link;
} /* if processing aggregate = FALSE */
/* process the master aggregate last */
else
{
    /* total the subordinate charges into the */
    /* totals record for this aggregate account */
    totals.subordinate_home =
aggregate_totals->aggregate_totals.subordinate_home;
    totals.subordinate_foreign =
aggregate_totals->aggregate_totals.subordinate_foreign;

    /* point back to the start of aggregate list */
    aggregate_totals_start = aggregate_totals;
    /* total the subordinate account charges */
    market_call_list->call_list =
        (struct call_struct *)NULL;
    market_call_list->alt_call_list =
        (struct call_struct *)NULL;
    /* copy the aggregate rate plan to current rate */
    /* plan record */
    copy_rate_plan(
        &aggregate_totals_start->rate_plan_rec,
        &customer_rate_plan);
    processing_aggregate = FALSE;
} /* else processing aggregate = TRUE */
} /* if master aggregate */
else if (cust_info_list->aggr ==
        AGGREGATE_SUBORDINATE)
{
    /* get the data for billing the subordinate from */
    /* the aggregate list */
    market_call_list->call_list =
        aggregate_totals_start->call_list;

    market_call_list->alt_call_list =
        aggregate_totals_start->alt_call_list;
    /* copy the aggregate rate plan to current rate */
    /* plan record */
    copy_rate_plan(
        &aggregate_totals_start->rate_plan_rec,
        &customer_rate_plan);
} /* if aggregate subordinate */
else
{
mark_time(2,mark_time_arr,1);
    market_call_list->call_list =
        ret_call_info(cust_info_list->cust_nr,
            market_rec.market_sid,
            &prorate_from_date,&bill_date,
            &(bill_info_rec.detail_sort_cd),
            &(market_call_list->alt_call_list));
}

```

```
mark_time(2,mark_time_arr,2);
```

```
if (!get_cust_rate_plan(&bill_info_rec,
    rate_plan_list,
    cust_info_list->cust_status,
    cust_info_list->cust_nr,
    &customer_rate_plan,
    &prorate_from_date,
    &prorate_to_date,
    &market_rec.bill_date,
    &activation_date,
    &deactivation_date,
    &suspend_date,
    &period_date,
    market_call_list->call_list,
    market_rec.init_pay_type,
    cust_info_list->nr_prorated_days))
{
    error_handler("bill_test",UNKNOWN,"no rate plan");
    error = TRUE;
} /* no rate plan */
} /* non aggregate */
```

```
taxer->getCustExemptions(&exemption_list,
    cust_info_list->cust_nr);
printf("Just Returned From getCustExempts for");
printf(" account number %10.10s\n",
    cust_info_list->cust_nr);
```

```
/* get the previous charge */
totals.previous_balance = bill_info_rec.current_chges;
```

```
/* get any A/R records or any adjustments */
mark_time(1,mark_time_arr,1);
ar_list = get_ar_info(cust_info_list->cust_nr,
    &total_non_call_totals,
    &bill_date);
```

```
adjustment_list =
    get_adj_info(cust_info_list->cust_nr,
        market_rec.market,
        &bill_date,
        cat_list,
        &bill_info_rec);
```

```
taxer->calcTax(adjustment_list,exemption_list,
    bill_date.date_str,
    cust_info_list->geo_code,
    bill_info_rec.service_class,
    cust_info_list->cust_nr,
    cust_info_list->city_resident);
taxer->buildTaxRegister(adjustment_list,
    &tax_register,
    cust_info_list->geo_code);
```

```
calc_ar_adj(ar_list,adjustment_list,&totals,
    cat_list,journal_list,&collect_adj_list,
    super);
```

```

mark_time(1,mark_time_arr,2);
/* account balance for aggregates is 0 */
if (cust_info_list->aggr == AGGREGATE_SUBORDINATE)
{
    totals.previous_balance = 0L;
    totals.unpaid = 0L;
}
else
    totals.unpaid = totals.previous_balance -
                    totals.payments;

/* calculate the rate plan charges - if any */
if (customer_rate_plan.rate_plan_id[0] != (char)NULL)
{
    taxer->calcTax(&customer_rate_plan,
                  exemption_list,bill_date.date_str,
                  cust_info_list->geo_code,
                  bill_info_rec.service_class,
                  cust_info_list->cust_nr,
                  cust_info_list->city_resident);
    taxer->buildTaxRegister(&customer_rate_plan,
                           &tax_register,
                           cust_info_list->geo_code);

    calc_rate_plan_charges(&customer_rate_plan,&totals,
                           journal_list);
}

/* calculate the recurring charge totals and debit */
/* the recurring charge balance - if appropriate */
/* NOTE: prorate from date is the activation date */
printf("BT no_active_days = %d\n",customer_rate_plan.no_active_days);

```

```

recur_list =
    get_recur_charges(cust_info_list->cust_nr,
                      cust_info_list->aggr,
                      &prorate_from_date,
                      &prorate_to_date,
                      &bill_date,
                      &market_rec.bill_date,
                      &deactivation_date,
                      &suspend_date,
                      &activation_date,

```



```

        market_rec.market,
        cust_info_list->cust_status,
        market_rec.init_pay_type,
        customer_rate_plan.no_active_days,
        cat_list,
        &bill_info_rec,
        cust_info_list->nr_prorated_days,
        misc_mkt_chg,
        market_rec.switch_name,
        cust_info_list->mobile_nr,
        super);

if (recur_list != (struct recur_struct *)NULL)
{

    taxer->calcTax(recur_list,exemption_list,
        bill_date.date_str,
        cust_info_list->geo_code,
        bill_info_rec.service_class,
        cust_info_list->cust_nr,
        cust_info_list->city_resident);
    taxer->buildTaxRegister(recur_list,
        &tax_register,
        cust_info_list->geo_code);
    calc_recur_charges(recur_list,&totals,
        journal_list);
} /* if recur_list */

/* calculate the nonrecurring charge totals */
nonrecur_list =
    get_nonrecur_charges(cust_info_list->cust_nr,
        market_rec.market,
        &bill_date,
        cat_list,
        &bill_info_rec);
if (nonrecur_list != (struct non_recur_struct *)NULL)
{

    taxer->calcTax(nonrecur_list,exemption_list,
        bill_date.date_str,
        cust_info_list->geo_code,
        bill_info_rec.service_class,
        cust_info_list->cust_nr,
        cust_info_list->city_resident);
    taxer->buildTaxRegister(nonrecur_list,
        &tax_register,
        cust_info_list->geo_code);
    calc_nonrecur_charges(nonrecur_list,&totals,
        journal_list);
} /* if nonrecur_list */

/* calculate the air time charges */
mark_time(3,mark_time_arr,1);
/* don't calculate airtime charges or roamer */
/* charges for master aggregates */
if (cust_info_list->aggr != AGGREGATE_MASTER)
{
    if (customer_rate_plan.rate_plan_id[0] !=
        (char)NULL)
    {

```

```

        taxable_calls = (call_struct *)NULL;

        market_call_list->airtime_tot =
            calc_call_charges(&customer_rate_plan,
                market_call_list->call_list,
                &totals,
                &market_call_list->call_totals,
                &market_rec,
                toll_airtime_list,
                journal_list,
                cust_info_list->cust_status,
                &plan,&bill_info_rec,
                &taxable_calls,
                free_number_ptr);

        taxer->calcTax(taxable_calls,exemption_list,bill_date.date_str,
            cust_info_list->geo_code,bill_info_rec.service_class,
            cust_info_list->cust_nr,cust_info_list->city_resident);
        taxer->buildTaxRegister(taxable_calls,&tax_register,
            cust_info_list->geo_code);
        taxer->summarizeTax(taxable_calls,&market_call_list->call_totals.air_tax,
            &market_call_list->call_totals.land_tax);
        // this assumes that the taxable calls has local,intra and inter calls
        // in that order.
        taxer->summarizeTax(taxable_calls,
            &toll_airtime_list->airtime_tax[MAX_ROAMER_TYPES],NULL);
        call_struct *iter = taxable_calls;
        taxer->addTax(&toll_airtime_list->local_access_tax[MAX_ROAMER_TYPES],
            iter->land_tax);
        iter = iter->link;
        taxer->addTax(&toll_airtime_list->intrastate_tax[MAX_ROAMER_TYPES],
            iter->land_tax);
        iter = iter->link;
        taxer->addTax(&toll_airtime_list->interstate_tax[MAX_ROAMER_TYPES],
            iter->land_tax);
        /* update airtime and tax data to call_info */
    }

mark_time(3,mark_time_arr,2);

    /* retrieve all roamer call records */
    /* NOTE: prorate from date is activation date */
    if (!ret_roamer_info(cust_info_list->cust_nr,
        market_call_list,
        market_rec.market_sid,
        cust_info_list->activation_date,
        &bill_date,
        toll_airtime_list,
        &(bill_info_rec.detail_sort_cd)))

for (market_call_struct *mc_iter = market_call_list->link; mc_iter;
    mc_iter = mc_iter->link)
{
    taxer->calcTax(mc_iter->call_list,exemption_list,bill_date.date_str,
        cust_info_list->geo_code,bill_info_rec.service_class,
        cust_info_list->cust_nr,cust_info_list->city_resident);
    taxer->buildTaxRegister(mc_iter->call_list,&tax_register,
        cust_info_list->geo_code);
}

        calc_roamer_charges(market_call_list,&totals,
            toll_airtime_list);
    } /* if not master aggregate */

```

```

/* total all charges by category */
total_charges(&totals,market_call_list);

/* set the detailed billing flag */
detail_key(&bill_info_rec,recur_list);
/* if there are no current or unpaid charges */
/* then do not print a bill - flag the customer */
/* as having no current or unpaid charges */
/* print the bill */
mark_time(4,mark_time_arr,1);

/* change to use freopen for subsequent opens */
if(((reopen_flag) &&
((tpfp = fopen(print_tmp_fn,"w+")) != NULL) &&
((tbdfp = fopen(bill_image_tmp_fn,"w+")) != NULL))
||
((tpfp = freopen(print_tmp_fn,"w+",tpfp)) != NULL) &&
((tbdfp = freopen(bill_image_tmp_fn,"w+",tbdfp)) != NULL))
{
    reopen_flag=TRUE;
    setvbuf(tpfp,pfile_buf_tmp,_IOFBF,153600);
    setvbuf(tbdfp,bfile_buf_tmp,_IOFBF,153600);
    init_bill(&bp,80,66,tpfp);
    init_bill(&dbp,80,66,tbdfp);

/* collect dunning information applicable. */
    get_dunning_data(&market_rec.bill_date,
                    cust_info_list,
                    &bill_info_rec,
                    &dunning_cust,
                    &cur_charge_list,
                    &totals,
                    &collect_adj_list,
                    &customer_rate_plan,
                    ddl_list,
                    &today's_date,
                    super);

    if ((cust_info_list->aggr != AGGREGATE_SUBORDINATE) &&
        (cust_info_list->aggr != WALK_IN))
    {
        switch(dunning_cust.treatment_notice)
        {
            case NO_TREATMENT:

printf("NO TREATMENT\n");
/* Compute balance anyway but won't get notice. (print_bill handles that) */
                standardDunning(&dunning_cust,
                                market_rec.leeway_amount);

                break;
            case STANDARD_TREAT:

printf("STANDARD\n");
/* Use standard treatment algorithm. */
                standardDunning(&dunning_cust,
                                market_rec.leeway_amount);

                break;
            case SPECIAL_TREAT:

printf("SPECIAL\n");
/* Use corporate treatment algorithm. */
                specialDunning(&dunning_cust,
                                market_rec.leeway_amount);

                break;
            case DEAL_TREAT:

printf("DEAL\n");
/* Use corporate treatment algorithm. */
                dealDunning(&dunning_cust,
                            market_rec.leeway_amount);

```

```

        break;
    case BAD_DEAL_TREAT:
        printf("BAD_DEAL\n");
        /* Use corporate treatment algorithm. */
        baddealDunning(&dunning_cust,
                      market_rec.leeway_amount);

        break;
    default: -

printf("DEFAULT\n");
/* This may happen given our screwy data security. So log and fix as needed. */
    error_handler("bill_test", UNKNOWN,
                  "Undefined dunning treatment code");
    error = TRUE;
    break; /* Just for the hell of it. */
} /* Balance based on account's treatment code */

printf("PAST DUE Account = %10.10s past due = %ld notice level = %c\n",
      dunning_cust.acct_nr,
      dunning_cust.past_due_balance,
      dunning_cust.notice_level);

/* catalog dunning action in statistics record. */
    acc_dunning_stats(&dunning_cust
                    &dunning_stats_hdr,
                    &dunning_stats);

/* Calculate a latefee */
//*****

late_fee_struct lfs;

lfs.market = &market_rec;
lfs.cust_info_list = cust_info_list;

lfs.dunning_cust = &dunning_cust;
lfs.bill_info_rec = &bill_info_rec;
lfs.cur_charge_list = cur_charge_list;
lfs.adjustment_list = &adjustment_list;
lfs.collect_adj_list = &collect_adj_list;
lfs.totals = &totals;
lfs.todays_date = &todays_date;
lfs.latefee_date = &latefee_date;

lfs.cat_list = cat_list;
lfs.ddl_list = ddl_list;
lfs.jrnl_list = journal_list;
lfs.exemptions = exemption_list;
//*****
        if(calc_latefee(&lfs, super))
        {
            error_handler("bill_test", UNKNOWN,
                          "Error calculating late fee.");
            error = TRUE;
        }
    else
    {
        /* Check for dunning exceptions */
        if(dunning_cust.notice_level != FYI_MESSAGE)
        {
            if(dunning_cust.notice_level == ERROR_NOTICE)
            {

```

```

        error_handler("bill_test",UNKNOWN,
        "Undefined notice level in bill_info");
        error = TRUE;
    }/* Fatal error invalid notice */
else
{
    dunning_exception(&dunning_cust,
        &dunning_exception_list,
        &dunning_stats_hdr);

    if(!commentLevels(&dunning_cust,
        &bill_date,
        &today's_date,
        market_rec.market,
        super))
    {
        error_handler("bill_test",UNKNOWN,
        "Error inserting late notice comment.");
        error = TRUE;
    }
    }/* else no error notice */
    }/* fyi's don't count here */
    }/* else no error latefee */

    if(update_bill_info(&bill_date,&dunning_cust,
        bill_info_rec.rowid))
    {
        error_handler("bill_test",UNKNOWN,
        "Error updating aged_analysis in bill_info");
        error = TRUE;
    }

    }/* Aggregates subordinates don't have balances*/
else
{
    dunning_cust.notice_level = FYI_MESSAGE;
    }/* Give subordinates FYI */

```

//*****

```

print_bill_struct pbs;
pbs.cust_info_rec = cust_info_list;
pbs.market_call_list = market_call_list;
pbs.totals = &totals;
pbs.recur_list = recur_list;
pbs.nonrecur_list = nonrecur_list;
pbs.ar_list = ar_list;
pbs.adjustment_list = adjustment_list;
pbs.mkt_rec = &market_rec;
pbs.bill_info_rec = &bill_info_rec;
pbs.rate_plan_rec = &customer_rate_plan;
pbs.tod_desc_list = tod_desc_list;
pbs.fyi_messages = fyi_messages;
pbs.airtime_tod_totals = market_call_list->airtime_tot;
pbs.rate_plan_prorate = customer_rate_plan.sc_pro_rate;
pbs.aggregate_totals = aggregate_totals_start;
pbs.display_date = &bill_date;
pbs.period_display_date = &period_date;
pbs.offset_display_date = &offset_display_date;
pbs.due_date = &due_date;
pbs.bp = &bp;
pbs.dbp = &dbp;
pbs.cat_list = cat_list;

```

```

pbs.airtime_detail_start = &airtime_detail_start;
pbs.todays_date = &todays_date;
pbs.dunning_cust = &dunning_cust;
//.....

    if(print_bill(&pbs,super))
    {
        error_handler("bill_test",UNKNOWN,
                      "printing bill");
        error = TRUE;
    } /* if print_bill */

//
//
    if((cust_info_list->aggr) == AGGREGATE_MASTER) &&
        (cust_info_list->aggr != WALK_IN))

/* See if this is a zero bill customer */
if(cust_info_list->aggr != WALK_IN)
{
    send_bill = check_zero_bill(&dunning_cust,
                                cust_info_list,
                                &dunning_stats_hdr,
                                &totals,
                                market_call_list,
                                &zero_bill_list,
                                &collect_adj_list,
                                bill_info_rec.pull_bill,
                                super);
}
else
{
    send_bill = TRUE;
}

/* Get number of pages generated for this bill */
if(send_bill)
dunning_stats_hdr.bill_pages +=
(bp.page_count + dbp.page_count) *
bill_info_rec.bill_copies;

    build_bill_detail(market,cust_info_list,
                      &bill_date,airtime_detail_start,
                      &bill_info_rec,pfp,bdfp,&bp,
                      &dbp,send_bill,
                      &dunning_stats_hdr);

/* close the print files */
fclose(tpfp);
fclose(tbdfp);
} /* fopen or freopen */
else
{
    printf("error opening bill print files\n");
    error = TRUE;
} /* fopen error */

mark_time(4,mark_time_arr,2);
/* build the commission_waivers report line */
mark_time(6,mark_time_arr,1);
    build_comw_rpt(&comw_rpt,
                  &comw_rpt_struct,
                  adjustment_list,
                  cust_info_list,
                  exemption_list,
                  todays_date.date_str,
                  &comw_list,

```

```

        bill_info_rec.service_class,
        &comw_amt_totals,
        &comw_fed_totals,
        &comw_state_totals,
        &comw_county_totals,
        &comw_loc_totals,
        parallel);

/* accumulate phone sales report */
acc_phone_sales(phone_sales_list, recur_list,
                nonrecur_list, cust_info_list);

/*endif

phone_sales_list_header_cur = phone_sales_list_header;
acc_phone_sales(phone_sales_list_header_cur->sales_list,
                recur_list,
                nonrecur_list,
                cust_info_list,
                temp_bill_params->ph_sales_jrnl_acct);
phone_sales_list_header_cur = phone_sales_list_header_cur->link;
acc_phone_sales(phone_sales_list_header_cur->sales_list,
                recur_list,
                nonrecur_list,
                cust_info_list,
                temp_bill_params->equip_sales_jrnl_acct);

/* Get copy of charge totals record for current
   charges table update */
add_totals(&totals, &current_charge_totals);

/* accumulate revenue by charge report */
acc_rev_chg(&rev_list, &recur_list,
            &nonrecur_list, &bill_info_rec,
            totals.monthly_access);

/* accumulate the airtime summary report totals */
if (customer_rate_plan.rate_plan_id[0] != (char)NULL)
    if (acc_airtime_summary(airtime_summary,
                            market_call_list->airtime_tot,
                            customer_rate_plan.rate_plan_id,
                            totals.monthly_access))
    {
        printf("airtime summary report error\n");
    } /* else acc_airtime_summary error */

mark_time(6, mark_time_arr, 2);
mark_time(7, mark_time_arr, 1);
/* update summary of cust activity */
upd_summary_list(
    cust_info_list->cust_nr,
    market,
    market_call_list,
    &totals,
    bill_date.date_str);

mark_time(7, mark_time_arr, 2);

memcpy(prev_acct_nr, cust_info_list->acct_nr, 10);

/* total the aggregate accounts */
if (cust_info_list->aggr == AGGREGATE_SUBORDINATE)
{
    /* copy the aggregate totals data into the */

```

```

/* aggregate totals list */
add_totals(&totals,
    &aggregate_totals_start->aggregate_totals);
add_call_totals(&market_call_list->call_totals,
    &aggregate_totals_start->aggregate_call_totals);
total_sub_aggr(aggregate_totals,
    aggregate_totals_start,
    market_call_list);
aggregate_totals_start =
    aggregate_totals_start->link;
cust_info_list = cust_info_list->link;
} /* if aggregate subordinate */
/* total all the subordinate charges for the */
/* current master account. this will allow */
/* correct reporting based on account number */
else if (cust_info_list->aggr == AGGREGATE_MASTER)
{
    /* pass the head of the aggregate list */
    total_aggregate(aggregate_totals_start,&totals,
        &market_call_list->call_totals);
} /* if master aggregate */

/* if this is the last aggregate then process the */
/* master aggregate last */
if (processing_aggregate &&
    memcmp(prev_acct_nr,cust_info_list->acct_nr,10))
    cust_info_list = master_aggregate_ptr;

} /* if build_market_call_list */
else
{
    error_handler("bill_test",UNKNOWN,
        "building market call list");
    error = TRUE;
} /* else build_market_call_list error */

/* update the number of prorated days
if (update_nr_prorated_days(cust_info_list->cust_nr))
{
    error_handler("bill_test",UNKNOWN,
        "update nr prorated days");
    error = TRUE;
} if update nr prorated days */

```

```

if((processing_aggregate) && ( cust_info_list->aggr != AGGREGATE_MASTER))
{

```

```

    /* call related charges */
    while (market_call_list !=
        (struct market_call_struct *)NULL)
    {
        /* free the subordinate lists */
        /* call list */
        while (market_call_list->call_list !=
            (struct call_struct *)NULL)
        {
            taxer->freeTaxList(
                &market_call_list->call_list->air_tax);
            taxer->freeTaxList(
                &market_call_list->call_list->land_tax);
            temp_list_start =
                (char *)market_call_list->call_list->link;
            free((char *)market_call_list->call_list);
            market_call_list->call_list =
                (struct call_struct *)temp_list_start;

```



```

    } /* while elements in list */

    /* Free call taxes */
    taxer->freeTaxList(
        &market_call_list->call_totals.air_tax);
    taxer->freeTaxList(
        &market_call_list->call_totals.land_tax);

/* airtime totals */
while (market_call_list->airtime_tot !=
    (struct airtime_totals *)NULL)
{
    temp_list_start =
        (char *)market_call_list->airtime_tot->link;
    free((char *)market_call_list->airtime_tot);
    market_call_list->airtime_tot =
        (struct airtime_totals *)temp_list_start;
} /* while elements in list */

temp_list_start = (char *)market_call_list->link;
free((char *)market_call_list);
market_call_list =
    (struct market_call_struct *)temp_list_start;
} /* while elements in list */

/* Free taxable calls list */
while (taxable_calls !=
    (struct call_struct *)NULL)
{
    taxer->freeTaxList(&taxable_calls->air_tax);
    taxer->freeTaxList(&taxable_calls->land_tax);
    temp_list_start =
        (char *)taxable_calls->link;
    delete taxable_calls;
    taxable_calls =
        (struct call_struct *)temp_list_start;
} /* while elements in list */

/* recurring charges */
while (recur_list != (struct recur_struct *)NULL)
{
    temp_list_start = (char *)recur_list->link;
    taxer->freeTaxList(&recur_list->tax);
    free((char *)recur_list);
    recur_list = (struct recur_struct *)temp_list_start;
} /* while elements in list */

/* nonrecurring charges */
while (nonrecur_list != (struct non_recur_struct *)NULL)
{
    temp_list_start = (char *)nonrecur_list->link;
    taxer->freeTaxList(&nonrecur_list->tax);
    free((char *)nonrecur_list);
    nonrecur_list =
        (struct non_recur_struct *)temp_list_start;
} /* while elements in list */

} /* if processing_aggregate */

/* tax exemptions */
if (exemption_list != (struct exemption_info *)NULL)
{
    taxer->freeExemptionList(&exemption_list);
}

```

```

    }
} while (!error && processing_aggregate);

/* build the AR report line */
build_ar_rpt(&ar_rpt_struct, cust_info_list,
            &bill_info_rec, &totals,
            market_call_list);

/* build the customer detail report */


/* total market call and non call totals */
total_totals(&total_non_call_totals,
            &total_call_totals,
            &total_roamer_totals,
            &totals,
            market_call_list);

/* add any unpaid charges or credit to the */
/* current charge and update the billing table*/

if (bill_commit && update_current_charges(cust_info_list,
            cur_charge_list,
            &current_charge_totals,
            bill_date.date_str, &collect_adj_list))
{
    error_handler("bill_test", UNKNOWN,
                "updating charge bill");

    error = TRUE;
} /* if error update current charge */
} /* if get current charges */

} /* if get_bill_info */
else
{
    error_handler("bill_test", UNKNOWN,
                "getting bill info");

    error = TRUE;
} /* else get_bill_info.error */

if (!bill_commit)
{
    EXEC SQL ROLLBACK;
}
else if (!error)
{
    EXEC SQL COMMIT;
}

/* free the customer associated linked lists */

```

```

/* Free aggregate totals list */
aggregate_totals = aggregate_totals_start;
while(aggregate_totals !=
      (struct aggregate_struct *)NULL)
{
    taxer->freeTaxList(
&aggregate_totals->aggregate_totals.noncall_tax);
    taxer->freeTaxList(
&aggregate_totals->aggregate_totals.payment_adj_tax);
    taxer->freeTaxList(
&aggregate_totals->aggregate_totals.home_adj_tax);
    taxer->freeTaxList(
&aggregate_totals->aggregate_totals.foreign_adj_tax);

    taxer->freeTaxList(
&aggregate_totals->aggregate_call_totals.air_tax);
    taxer->freeTaxList(
&aggregate_totals->aggregate_call_totals.land_tax);

    aggregate_totals_start = aggregate_totals->link;
    free((char *)aggregate_totals);
    aggregate_totals = aggregate_totals_start;
}/* while aggregate struct nodes */

/* free rate plan taxes */
taxer->freeTaxList(&customer_rate_plan.tax);

/* Free taxable calls list */
while (taxable_calls !=
      (struct call_struct *)NULL)
{
    taxer->freeTaxList(&taxable_calls->air_tax);
    taxer->freeTaxList(&taxable_calls->land_tax);
    temp_list_start =
        (char *)taxable_calls->link;
    delete taxable_calls;
    taxable_calls =
        (struct call_struct *)temp_list_start;
} /* while elements in list */

/* current charges */
while (cur_charge_list !=
      (struct cur_charge_struct *)NULL)
{
    temp_list_start = (char *)cur_charge_list->link;
    free((char *)cur_charge_list);
    cur_charge_list =
        (struct cur_charge_struct *)temp_list_start;
} /* while elements in list */

/* ar */
while (ar_list != (struct ar_struct *)NULL)
{
    temp_list_start = (char *)ar_list->link;
    free((char *)ar_list);
    ar_list = (struct ar_struct *)temp_list_start;
} /* while elements in list */

/* adjustment list copy */
while (collect_adj_list !=
      (struct collect_adj_struct *)NULL)
{
    temp_list_start = (char *)collect_adj_list->link;
    free((char *)collect_adj_list);
    collect_adj_list =

```

```

        (struct collect_adj_struct *)temp_list_start;
    } /* while elements in list */

/* adjustments */
while (adjustment_list !=
        (struct adjustment_struct *)NULL)
{
    temp_list_start = (char *)adjustment_list->link;
    taxer->freeTaxList(&adjustment_list->tax);
    free((char *)adjustment_list);
    adjustment_list =
        (struct adjustment_struct *)temp_list_start;
} /* while elements in list */

/* recurring charges */
while (recur_list != (struct recur_struct *)NULL)
{
    temp_list_start = (char *)recur_list->link;
    taxer->freeTaxList(&recur_list->tax);
    free((char *)recur_list);
    recur_list = (struct recur_struct *)temp_list_start;
} /* while elements in list */

/* nonrecurring charges */
while (nonrecur_list != (struct non_recur_struct *)NULL)
{
    temp_list_start = (char *)nonrecur_list->link;
    taxer->freeTaxList(&nonrecur_list->tax);
    free((char *)nonrecur_list);
    nonrecur_list =
        (struct non_recur_struct *)temp_list_start;
} /* while elements in list */

/* call related charges */
while (market_call_list !=
        (struct market_call_struct *)NULL)
{
    /* free the subordinate lists */
    /* call list */
    while (market_call_list->call_list !=
            (struct call_struct *)NULL)
    {
        taxer->freeTaxList(
            &market_call_list->call_list->air_tax);
        taxer->freeTaxList(
            &market_call_list->call_list->land_tax);
        temp_list_start =
            (char *)market_call_list->call_list->link;
        free((char *)market_call_list->call_list);
        market_call_list->call_list =
            (struct call_struct *)temp_list_start;
    } /* while elements in list */

    /* Free call taxes */
    taxer->freeTaxList(
        &market_call_list->call_totals.air_tax);
    taxer->freeTaxList(
        &market_call_list->call_totals.land_tax);

    /* airtime totals */
    while (market_call_list->airtime_tot !=
            (struct airtime_totals *)NULL)
    {
        temp_list_start =
            (char *)market_call_list->airtime_tot->link;
        free((char *)market_call_list->airtime_tot);
    }
}

```

```

        market_call_list->airtime_tot =
            (struct airtime_totals *)temp_list_start;
    } /* while elements in list */

    temp_list_start = (char *)market_call_list->link;
    free((char *)market_call_list);
    market_call_list =
        (struct market_call_struct *)temp_list_start;
    } /* while elements in list */

/* if aggregate account free all members of the */
/* account */
do
{
    memcpy(prev_acct_nr,cust_info_list->acct_nr,10);
    temp_list_start = (char *)cust_info_list->link;
    free((char *)cust_info_list);
    cust_info_list =
        (struct cust_struct *)temp_list_start;
    } while (cust_info_list != (cust_struct *)NULL &&
        !memcmp(cust_info_list->acct_nr,
            prev_acct_nr,10));

mark_time(0,mark_time_arr,2);
    memcpy(shmaddress,&seg_perf,
        sizeof(struct seg_perf_struct));
    } /* while cust_info_list */

if (!error)
{
    if (!parallel)
    {
        printf("BUILDING THE REPORTS\n");
        /* add the totals to the accounts receivable report */
        add_ar_totals(&ar_rpt_struct,
            &total_non_call_totals,
            &total_call_totals,
            &total_roamer_totals);

        /* build the airtime summary report */
        build_as_rpt(&as_rpt,&as_rpt_struct,airtime_summary,
            tod_desc_list);

        /* build the toll airtime summary report */
        build_tas_rpt(&tas_rpt,&tas_rpt_struct,
            toll_airtime_list);

        /* build the billing report */
        build_bill_rpt(&billing_rpt,&billing_rpt_struct,
            &total_non_call_totals,
            &total_call_totals,
            &total_roamer_totals);

        /* build the journal summary report */
        build_js_rpt(&js_rpt,&js_rpt_struct,journal_list,
            &total_non_call_totals,&total_call_totals,
            &total_roamer_totals,super);

        /* build phone sales report */
        build_ps_rpt(&ps_rpt,&ps_rpt_struct,phone_sales_list);
        build_ps_rpt(&ps_rpt,&ps_rpt_struct,phone_sales_list_header);
    }
}

```

```

/* build the tax register report */
build_tr_rpt(&tr_rpt,&tr_rpt_struct,tax_register);

/* add commission waivers totals */
add_comw_totals(&comw_rpt,&comw_rpt_struct,
               comw_amt_totals,comw_fed_totals,
               comw_state_totals,comw_county_totals,
               comw_loc_totals);

} /* if !parallel */
else
{
mark_time(13,mark_time_arr,1);
/*****/
rpt_data_struct rds;
rds.segment = segment;
rds.bill_date = bill_date.date_str;
rds.market = market;
rds.total_call_totals = &total_call_totals;
rds.total_non_call_totals = &total_non_call_totals;
rds.total_roamer_totals = &total_roamer_totals;
rds.airtime_summary = airtime_summary;
rds.tod_desc_list = tod_desc_list;
rds.toll_airtime_list = toll_airtime_list;
rds.journal_list = journal_list;
rds.phone_sales_list = phone_sales_list_header;
rds.tax_register = tax_register;
rds.rev_list = rev_list;
rds.comw_list = comw_list;
rds.comw_amt_totals = comw_amt_totals;
rds.comw_fed_totals = comw_fed_totals;
rds.comw_state_totals = comw_state_totals;
rds.comw_county_totals = comw_county_totals;
rds.comw_loc_totals = comw_loc_totals;
rds.dunning_exception_list = dunning_exception_list;
rds.zero_bill_list = zero_bill_list;
rds.discount_plans = &plan;
/*****/
error = ins_rpt_data(&rds);

if(error)
{
error_handler("bill_test",UNKNOWN,
"Report data insert had error(s).");
}

```

```

mark_time(13,mark_time_arr,2);
        /* Insert report stuff into database */
        /* if !error */
        else
        {
            error_handler("bill_test",UNKNOWN,
*WARN: Report data will not be inserted due to previous error.*);
            error = TRUE;
        }
        /* if build toll airtime list */
        else
        {
            error_handler("bill_test",UNKNOWN,
                        "building toll airtime list");

            error = TRUE;
        } /* else get_cust_list error */

        /* if fopen report files */
        else
        {
            error_handler("bill_test",FILEOPEN,"report files");
            error = TRUE;
        } /* else fopen report files error */

        /* if get_cust_list */
        else
        {
            error_handler("bill_test",UNKNOWN,"getting customer list");
            error = TRUE;
        } /* else get_cust_list error */
        /* if get_print_cat */
        else
        {
            error_handler("bill_test",UNKNOWN,"getting print category list");
            error = TRUE;
        } /* else error getting print_cat info */
        /* if get_tod_desc_list */
        else
        {
            error_handler("bill_test",UNKNOWN,"getting tod description list");
            error = TRUE;
        } /* else get_tod_desc_list error */

        /* if get_date_values */
        else
        {
            error_handler("bill_test",UNKNOWN,"getting date values");
            error = TRUE;
        } /* else get_date_values error */
        /* if get_rate_list */
        else
        {
            error_handler("bill_test",UNKNOWN,"getting rate list data");
            error = TRUE;
        } /* else get_rate_list error */

        /* if get leeway amount */
        else
        {

```

```

    error_handler("bill_test",UNKNOWN,"getting leeway amount");
    error = TRUE;
} /* else get_rate_list error */
} /* if get due date list*/

else
{
    error_handler("bill_test",UNKNOWN,"getting due date list");
    error = TRUE;
} /* else get_due_list error */

} /* if get_market */
else
{
    error_handler("bill_test",UNKNOWN,"getting market information");
    error = TRUE;
} /* else error getting market information */

} /* if fopen */
else
{
    error_handler("bill_test",FILEOPEN,argv[2]);
    error = TRUE;
} /* else fopen error */

} /* if log on */
else
{
    printf("\ncan't log on to Oracle\n");
    error = TRUE;
} /* else - logon */

/* get the last bill date and update the market table */
/* with the current bill date */
if (bill_commit)
{
    printf("UPDATED BILL DATE\n");
    update_bill_date(&bill_date,&offset_display_date,&due_date,market);
}

if ((!parallel) && (!error))
{
    /* print the automatic reports */

    /* print the accounts receivable report
    print_report(ar_rpt,&ar_rpt_struct); */

    /* print the airtime summary report */
    print_report(as_rpt,&as_rpt_struct);

    /* print the toll and airtime summary report */
    print_report(tas_rpt,&tas_rpt_struct);

    /* print the billing report */
    print_report(billing_rpt,&billing_rpt_struct);

    /* print the journal summary report */
    print_report(js_rpt,&js_rpt_struct);

    /* print the phone sales report */
    print_report(ps_rpt,&ps_rpt_struct);

    /* print the tax register report */
    print_report(tr_rpt,&tr_rpt_struct);

    /* print the charge detail report */

```



```

/* print the commission waivers report */
print_report(comm_rpt,&comm_rpt_struct);

/* ----- */
/* - Report all data the was collected */
/* during the call discounting processing */
/* ----- */
if(discountReporting(&plan.market,bill_date.date_str) == -1)
{
    error_handler("Call Discounting",UNKNOWN,"Could not create report");
}

} /* if !parallel print reports */

if (error || !bill_commit)
{
    error=FALSE;
    printf("ROLLBACK\n");
    EXEC SQL ROLLBACK WORK;
    if (sqlca.sqlcode != NOT_SQL_ERROR)
    {
        error = TRUE;
        error_handler("rollback",ORACLESELECT,sqlca.sqlerrm.sqlerrmc);
    } /* if sql error */
} /* if error */

insert_dunning_activity(&market_rec,&bill_date,&due_date,&dunning_stats_hdr,
                        dunning_stats,segment);
EXEC SQL COMMIT WORK RELEASE;
if (sqlca.sqlcode != NOT_SQL_ERROR)
{
    error = TRUE;
    error_handler("commit",ORACLESELECT,sqlca.sqlerrm.sqlerrmc);
} /* if sql error */

/*****/
mark_time(5,mark_time_arr,2);
memcpy(shmaddress,&seg_perf: (sizeof(struct seg_perf_struct)));

sprintf(sxcp_file,"sxcp.rpt");
sprintf(dxcp_file,"dxcp.rpt");
sprintf(zero_file,"zero.rpt");
    if(!error) && (!parallel)
        && ((sxcp_rpt_struct.rpt_file =
            fopen(sxcp_file,"w+")) != NULL)
        && ((zero_rpt_struct.rpt_file =
            fopen(zero_file,"w+")) != NULL)
        && ((dxcp_rpt_struct.rpt_file =
            fopen(dxcp_file,"w+")) != NULL)
    {
        build_rev_rpt(rev_list,rev_rpt_struct.rpt_file,
                    bill_date.date_str,market,super);
    }

/* Build dunning exception rpt */
if(dunning_exception_list !=
    (struct collections_info *)NULL)
    build_exception_rpt(sxcp_rpt_struct.rpt_file,
                      dxcp_rpt_struct.rpt_file,
                      &dunning_exception_list,market,
                      bill_date.date_str,
                      temp_bill_params);

```

```

/* Build zero activity (no bill) rpt */
    if(zero_bill_list !=
        (struct zero_bill_struct *)NULL)
        build_zero_rpt(zero_rpt_struct.rpt_file,
                        &zero_bill_list,market,
                        bill_date.date_str,
                        temp_bill_params);
    /* Build reports if not aborting */

// free airtime_summary list
while (airtime_summary != (struct airtime_summary_struct *)NULL)
{
    // free airtime_totals list
    while (airtime_summary->airtime_tot != (struct airtime_totals *)NULL)
    {
        temp_list_start = (char *)airtime_summary->airtime_tot->link;
        //FCHECK(airtime_totals);
        free((char *)airtime_summary->airtime_tot);
        airtime_summary->airtime_tot =
            (struct airtime_totals *)temp_list_start;
    } /* while elements in list */

    temp_list_start = (char *)airtime_summary->link;
    //FCHECK(airtime_summary_struct);
    free((char *)airtime_summary);
    airtime_summary =
        (struct airtime_summary_struct *)temp_list_start;
} /* while elements in list */

// free bill detail sort code lookup table
get_sort_info(-1,"FREE");

// free memory used by tax interface and dump cache statistics
delete taxer;

/* close reallocated stdout */
if(!parallel)
{
    fclose(as_rpt_struct.rpt_file);
    fclose(tas_rpt_struct.rpt_file);
    fclose(js_rpt_struct.rpt_file);
    fclose(ps_rpt_struct.rpt_file);
    fclose(tr_rpt_struct.rpt_file);
    fclose(rev_rpt_struct.rpt_file);
    fclose(billing_rpt_struct.rpt_file);
}/* if not parallel mode, close sequential report files opened */

// fclose(sxcp_rpt_struct.rpt_file);
fclose(zero_rpt_struct.rpt_file);
// fclose(dxcp_rpt_struct.rpt_file);
fclose(ar_rpt_struct.rpt_file);
fclose(comw_rpt_struct.rpt_file);
fclose(fpstd);
fclose(fpstdc);
fclose(pfp);
fclose(bdfp);

/* for reporting exit status to parallel manager */
if(error) exit(1);
else exit(0);

} /* bill test */

```

```

void mark_time(int remark_nr,mark_struct *time_array,int mark_number)
// int    remark_nr; /* the remark number */
// struct mark_struct time_array[];
// int    mark_number;
{
    time_t curtime; /* time in seconds */
    struct tm *loc_time;
    static char last_account_nr[11] = "xxxxxxxxxx";
/*
    struct timeval tp; /* pointer to timeval struct in sys/time.h */
    struct timezone tzp; /* pointer to timezone struct in sys/time.h */
*/
    /* set the minutes west of Greenwich and timezone treatment */
    /* tzp.tz_minuteswest = 240; /* 4 hours west */
    tzp.tz_dsttime = 1; /* daylight savings applies appropriately */
*/
    if (curtime = time(0)) /* ptx change */
    /* if (!gettimeofday(&tp,&tzp)) */
    {
        loc_time = localtime(&curtime);
        /* determine the elapsed time since the last mark */
        if (mark_number == 1)
        {
            /* printf("%s %s",time_array[remark_nr].remark,ctime(&tp.tv_sec)); */
            printf("%s %s",time_array[remark_nr].remark,asctime(loc_time));
        }
        if (mark_number == 2)
        {
            printf("%s - time elapsed since last mark: secs %f\n",
                time_array[remark_nr].remark,
                (float)((float)curtime - (float)time_array[remark_nr].seconds));
        }
/* Multi-threaded segment performance statistics */
        if(remark_nr != 5)
        {
            seg_perf.last_cust_time = curtime - time_array[remark_nr].seconds;

            if(memcmp(seg_perf.last_account,last_account_nr,10) == 0)
            {
                seg_perf.last_acct_time += seg_perf.last_cust_time;
            }
            else
            {
                memcpy(last_account_nr,seg_perf.last_account,10);
                seg_perf.last_acct_time = seg_perf.last_cust_time;
            }

            if(seg_perf.slow_time < seg_perf.last_cust_time)
            {
                seg_perf.slow_time = seg_perf.last_cust_time;
            }
            else if(seg_perf.fast_time > seg_perf.last_cust_time)
            {
                seg_perf.fast_time = seg_perf.last_cust_time;
            }
            seg_perf.elapsed_time += seg_perf.last_cust_time;
        }
        else
        {
            seg_perf.total_time = curtime - time_array[remark_nr].seconds;
            seg_perf.running = 0;
            seg_perf.complete = 1;
        }
        /* ptx conversion */
    }
    time_array[remark_nr].seconds = curtime; /* ptx conversion */
}

```

